CS4125 System Analysis and Design

Lecturer: J. J. Collins

Assistant: Feng Chen

2015

AUDIOTRON

Fabio Francomacaro SID [illegible]

[illegible]

Andrew Cusack SID [illegible]

11/19/2015

Table of Contents

Narrative

Introduction

The International Federation of the Phonographic Industry (IFPI) (2015) states:

“The music business continues to expand into new markets and create new business models, attracting more users to digital music services and bringing artists to a wider global audience…

Subscription services, part of an increasingly diverse mix of industry revenue streams, are going from strength to strength. Revenues from music subscription services — including free-to-consumer and paid-for tiers — grew by 39.0 per cent in 2014 and growing consistently across all major markets…

Revenues from advertising-supported streaming services, such as YouTube and Vevo, are also growing — up 38.6 per cent in 2014.”

Methodology and research

The entire research necessary for the completion of this project has been performed on the Internet. Software used for the creation of graphs and diagrams include: Microsoft Word, Microsoft Paint, Cadifra UML Editor 1.3, Gliffy.com.

Audiotron

This document presents the proposed system design for a new client-server system referred hereafter as AUDIOTRON and designed to create a vibrant online community where artists can share and showcase their projects and have a chance to be recognized. While running Audiotron one can upload or download music files, create albums with themed playlists and share those albums with friends or other users sharing the same interests. While providing services for free, Audiotron will provide income for support and maintenance through advertising. Administrators are responsible for the approval of Advertisement campaigns, user uploaded content and maintenance and upgrading of the described system. Each user should be aware that their work, if unregistered, is available to the community for redistributions, remixes and tweaks. Audiotron recommends the Creative Commons Attribution licence.

Software Development Lifecycle

The Audiotron project will be developed adhering to the Agile Manifesto.

According to the US Department of Veteran Affairs (2015), Scrum Agile provides a flexible, iterative development where releases are generated after sprints of two to four weeks. Such methodology allows for refinement of requirements and design over the entire Software Development Lifecycle. Ideally each release should provide immediate feedback from users allowing for a tight cooperation between developers and stakeholders. This cooperation should result in a clear sense of project progress when compared with a more traditional waterfall approach. User Stories depicting both regular users and sponsors are used as the foundation of the functional design of the Audiotron system.

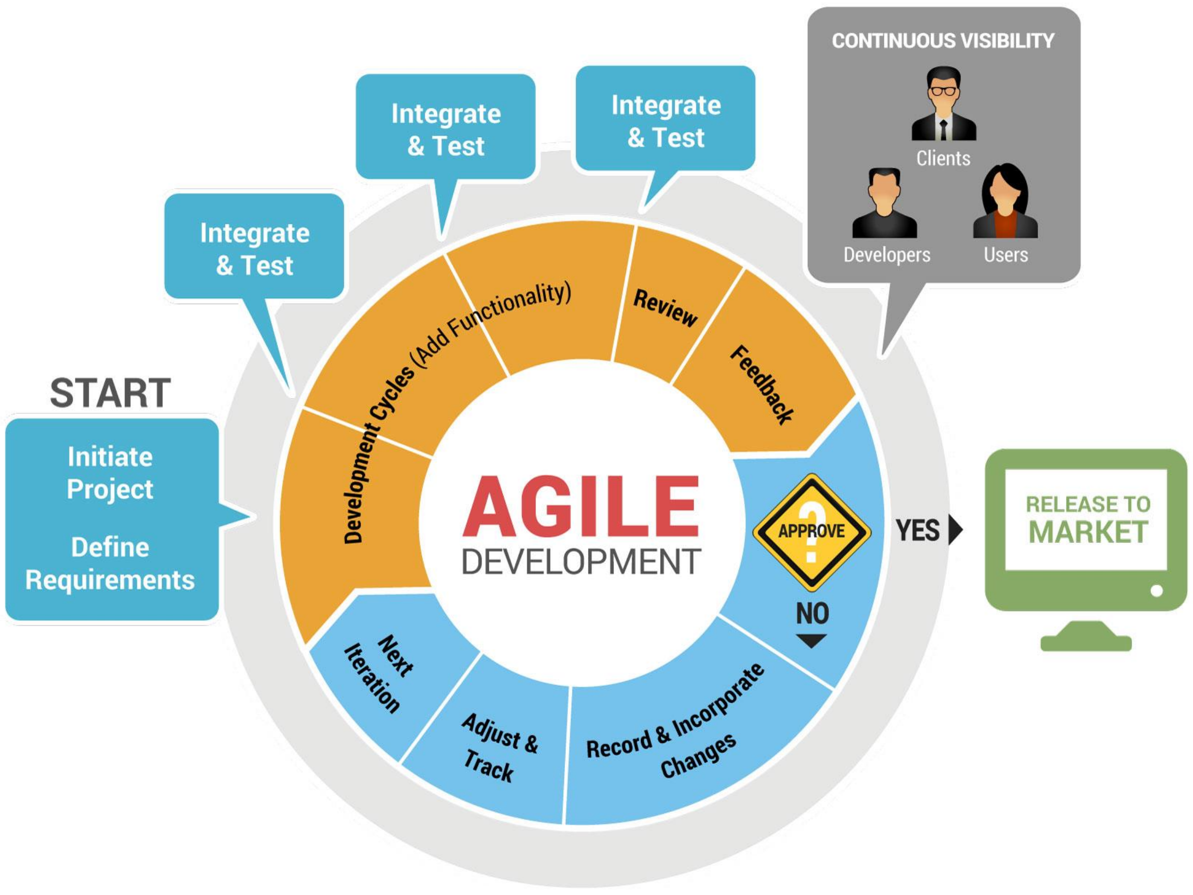


Illustration 1

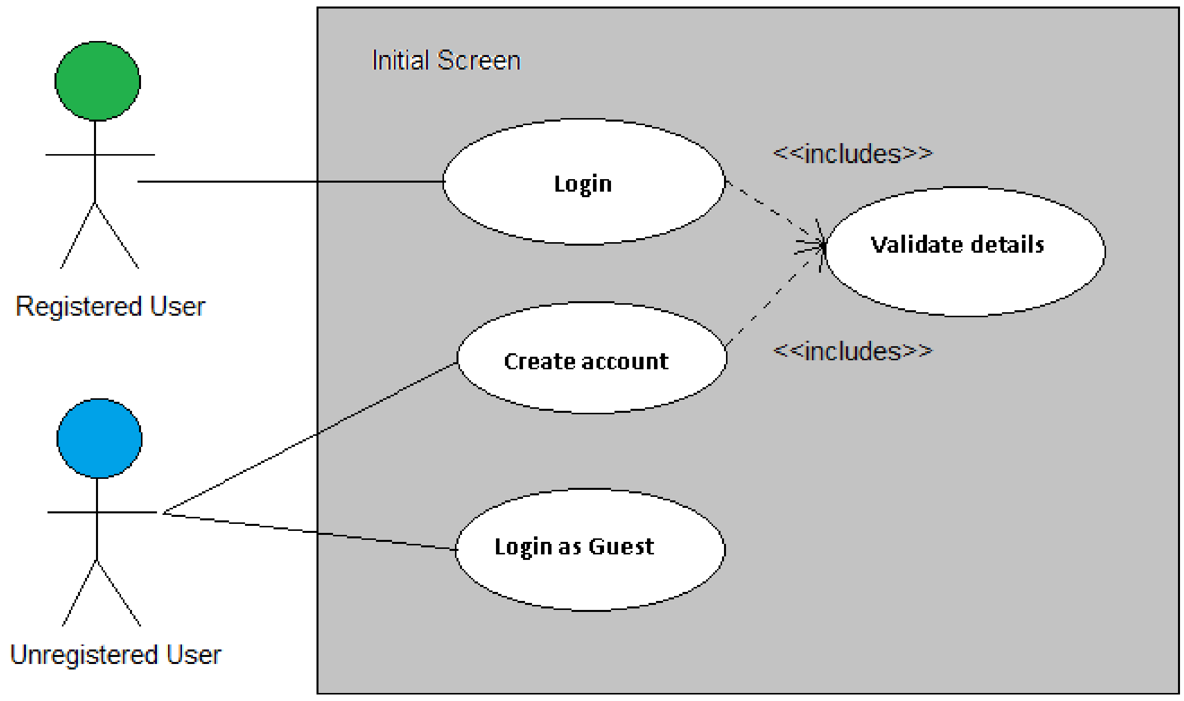
Project Plan and Allocation of Roles

Deliverable	Description	Student	Week
Presentation	Company Logo/ Cover Page Design	[redacted]	10
Narrative	Scenario Description	[redacted]	4
Software Lifecycle	Model of Software Development lifecycle	[redacted]	6
Project Plan	Specifying Milestones and roles	Achieved as a group	4
Requirements	Use case Diagrams Use case Descriptions Structured use case descriptions Non functional attributes Prototypes Screenshots	[redacted] [redacted] [redacted] [redacted] [redacted]	5 6 6 6 7
System Architecture	Architecture Diagram	[redacted]	7
Analysis Sketches	Identify Classes Class Diagram Communication Diagram E-R diagram	[redacted] [redacted] [redacted] [redacted]	8 8 8 8
Code	Implementation	[redacted]	10 to 12
Design	Architectural Diagram Class Diagram Interaction Diagram State chart Description of adopted design patterns patterns, approach to concurrency support	[redacted] [redacted] [redacted] [redacted] [redacted]	12 12 12 12 12
Critique	Critique on quality of analysis/design	[redacted]	12
References	Sources of information	[redacted]	12

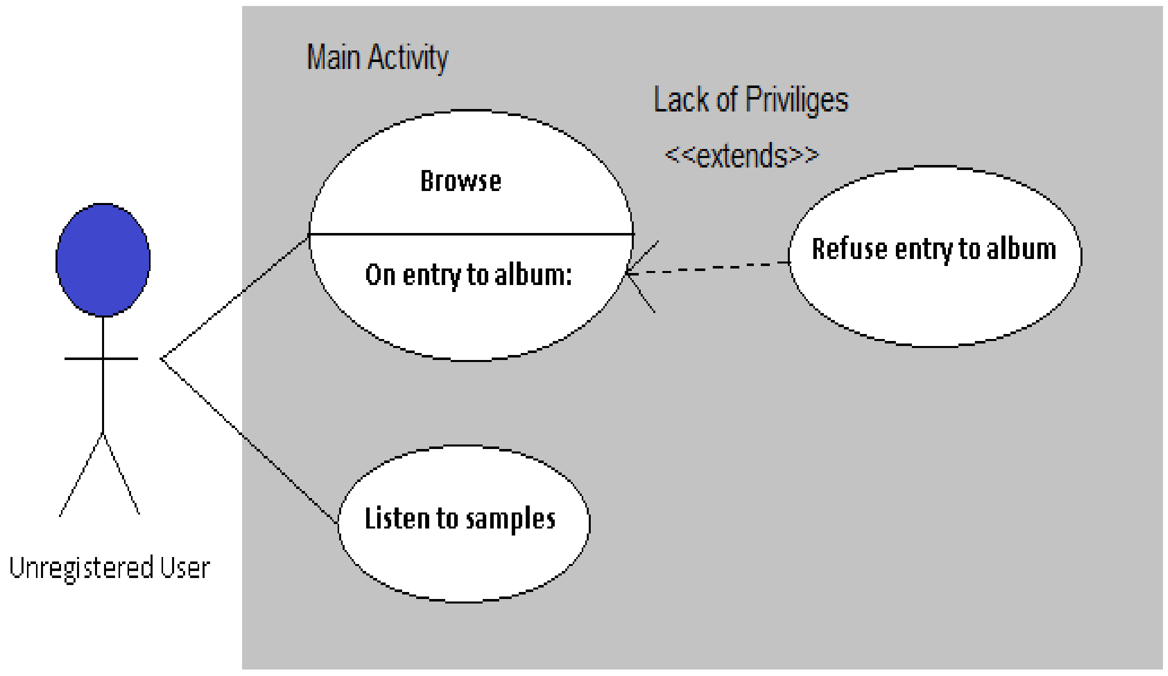
Requirements

Use Cases Diagrams

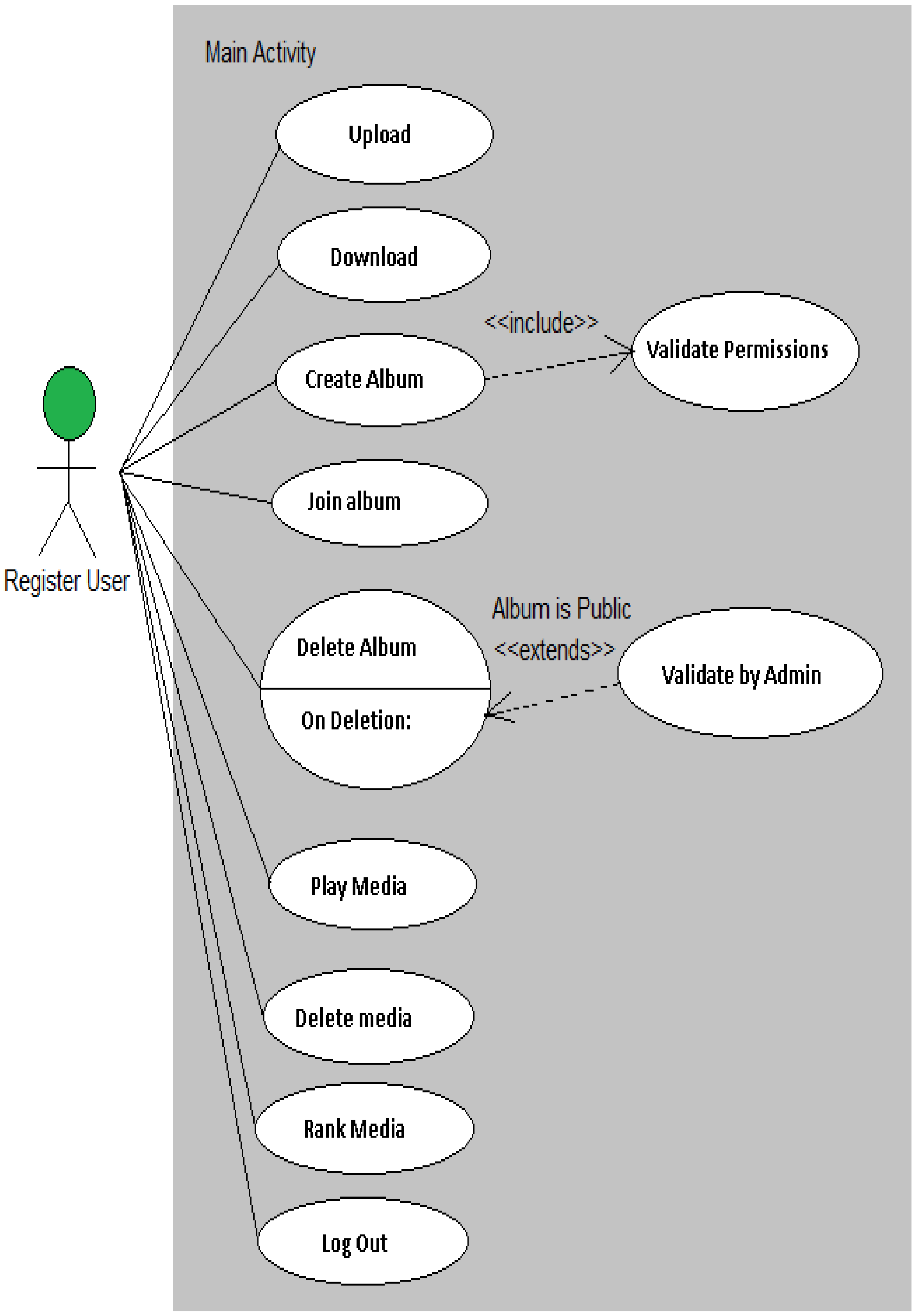
Initial Screen



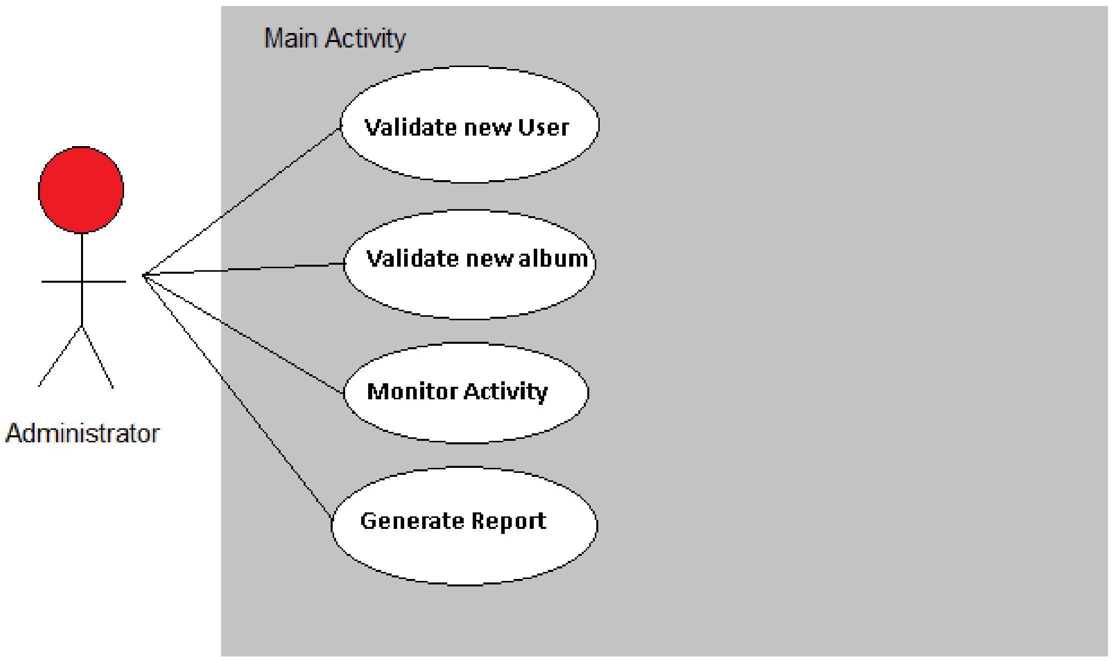
Main Activity / User



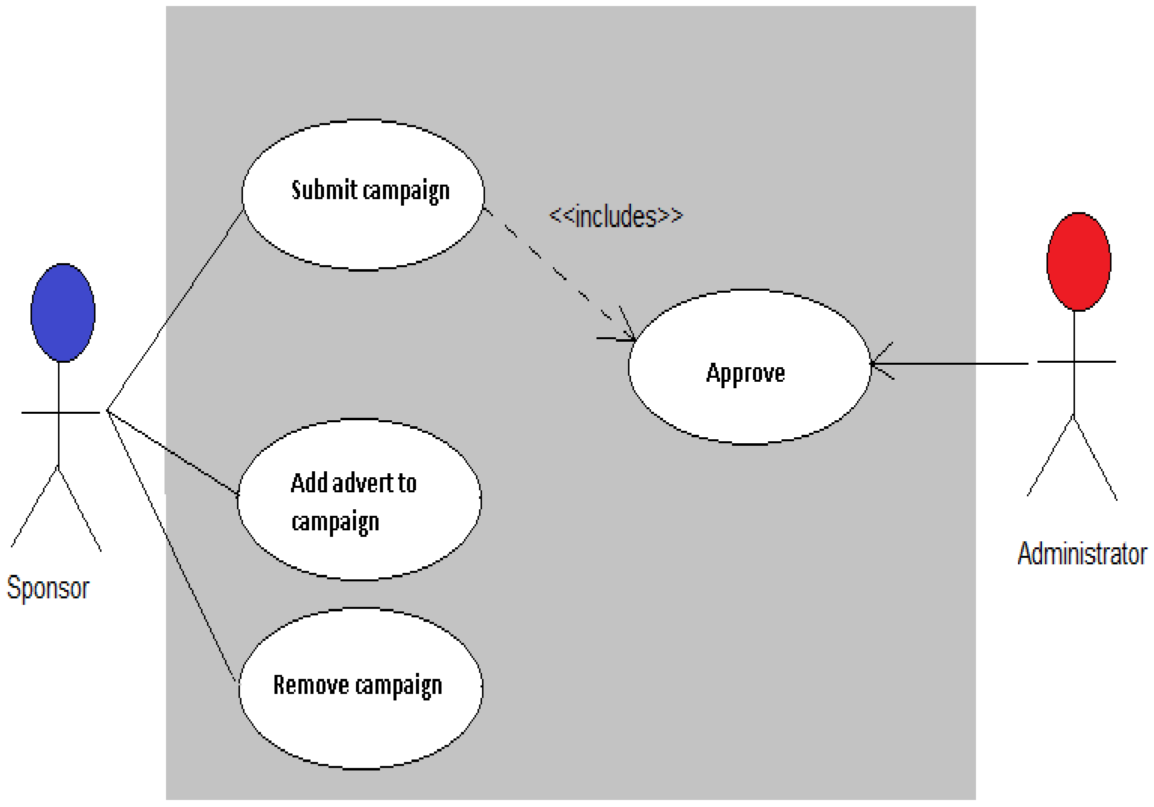
Main Activity / Registered User



Main Activity / Administrator



Business Tier Use Cases



Note: all of the above diagrams have been drawn with Microsoft Paint.

Use Cases Structured Description

Login

ACTOR'S ACTION	SYSTEM RESPONSE
1) User selects "Login" from Initial Screen.	2) Login form is displayed.
3) User enters username and password and clicks on "Login" button.	4) Login details are verified. Main Activity window is displayed.
ALTERNATIVE ROUTE	
3a) User enters wrong Login details.	4a) Login details cannot be verified. Error message is displayed to User and Login form is displayed. Invalid Login is logged for inspection.

Non Functional Requirements:

- Security: User details and personal information should be encrypted.
- Performance: User should get to Main Activity window after Login button press within 1 second.

Create Account

ACTOR'S ACTION	SYSTEM RESPONSE
1) User selects "Create Account" from Initial Screen.	2) Register form is displayed.
3) User enters Name, email address and desired password and clicks on "Register" button.	4) User details are verified. Main Activity window is displayed.
ALTERNATIVE ROUTE	
3a) User enters wrong email address.	4a) User details cannot be verified. Error message is displayed, attempt is logged for inspection and Register form is displayed.

Non Functional Requirements:

- Security: User details and personal information should be encrypted.
- Performance: Confirmation email should be sent within 5 seconds.

Login as Guest

ACTOR'S ACTION	SYSTEM RESPONSE
1) User selects "Login as guest" from Initial Screen.	2) Main Activity window is displayed.

Listen To Samples

ACTOR'S ACTION	SYSTEM RESPONSE
1) User selects media from the Main Activity screen.	2) User permissions and privileges are verified. Media file is played.
ALTERNATIVE ROUTE	
	2a) User does not have permissions to access the specified file. Error message is displayed, attempt is logged for inspection and main activity resumed.

Non Functional Requirements:

- Performance: Media should start playing within 5 seconds from request.

Browse

ACTOR'S ACTION	SYSTEM RESPONSE
1) User selects a custom created Album.	2) User permissions and privileges are verified. Album is open for browsing.
ALTERNATIVE ROUTE	
	2a) User does not have permissions to access the specified album. Error message is displayed, event logged for inspection and main activity resumed.

Upload

ACTOR'S ACTION	SYSTEM RESPONSE
1) User selects Upload Media.	2) User permissions and privileges are verified. Upload Media Form is displayed.
3) User fills form and clicks Upload.	4) File upload begins from user computer to server.
ALTERNATIVE ROUTE	
	2a) User does not have permissions to upload media. Error message is displayed, event logged and main activity resumed.
3a) User fills form incorrectly.	4a) Error message displayed, event logged. And Upload form displayed.
	4b) Error occurs during upload of file at server point. User is notified and Main Activity resumed.
5) Error occurs during transmission at User point.	6) Incident is logged for monitoring by administrator and user notified. Main Activity resumes.

Non Functional Requirements:

- Security: Secure connection should be established.
- Performance: Error risk should be at minimum.

Download

ACTOR'S ACTION	SYSTEM RESPONSE
1) User selects Download Media.	2) User permissions and privileges are verified. Download confirm form is displayed.
3) User agrees to download.	4) File download begins from server to user.
ALTERNATIVE ROUTE	
	2a) User does not have permissions to download media. Error message is displayed, event logged and main activity resumed.
	4a) Error occurs during download of file at Server point. User is notified and Main Activity resumed.
5) Error occurs during download at User point.	5) Incident is logged and user notified. Main Activity resumes.

Non Functional Requirements:

- Security: Secure connection should be established.
- Performance: Error risk should be at minimum.

Create Album

ACTOR'S ACTION	SYSTEM RESPONSE
1) User selects Create Album.	2) User permissions and privileges are verified. Create Album form is displayed.
3) User fills form and clicks Create Album.	4) Album is created.
ALTERNATIVE ROUTE	
	2a) User does not have permissions to create an album. Error message is displayed, event logged and main activity resumed.
3a) User fills form incorrectly.	4a) Error message displayed. Create Album form displayed.

Join Album

ACTOR'S ACTION	SYSTEM RESPONSE
1) User selects Join Album.	2) User permissions and privileges are verified. Album is added to user list.
ALTERNATIVE ROUTE	
	2a) User does not have permission to join the album. Error message is displayed and Main

	Activity resumed.

Non Functional Requirements:

- Usability: consider giving people the option of splitting an album as well.

Delete Album

ACTOR'S ACTION	SYSTEM RESPONSE
1) User selects Delete Album.	2) User ownership of album is verified. Album is removed from database.
ALTERNATIVE ROUTE	
	2a) User does not have ownership of the album. Error message is displayed and Main Activity resumed.

Play Media

ACTOR'S ACTION	SYSTEM RESPONSE
1) User selects Play.	2) Media file is played
ALTERNATIVE ROUTE	
	2a) User does not have ownership of the album. Error message is displayed and Main Activity resumed.

Delete Media

ACTOR'S ACTION	SYSTEM RESPONSE
1) User selects Delete Media.	2) User ownership of media is verified. Confirmation message is displayed.
3) User confirms deletion of file.	4) File is deleted. Main Activity is resumed.
ALTERNATIVE ROUTE	
	2a) User does not have ownership of the file. Error message is displayed and Main Activity resumed.

Rank Media

ACTOR'S ACTION	SYSTEM RESPONSE
1) User selects Rank Media.	2) Rank Form is displayed.
3) User fills rank form.	4) File is ranked. User comments are logged for inspection before being added to the comment

	list.
ALTERNATIVE ROUTE	
3a) User fills rank form incorrectly.	4a) Error message is displayed and Main Activity resumed.
	5) User comments are considered inappropriate and rank is not considered valid.

Non Functional Requirements:

- Security: consider banning user whose behaviour is considered inappropriate.

Log Out

ACTOR'S ACTION	SYSTEM RESPONSE
1) User selects Log out.	2) System logs user out. Initial Screen is displayed.

Non Functional Requirements:

- Security: System must ensure that User is fully Logged Out.

Validate new User

ACTOR'S ACTION	SYSTEM RESPONSE
1) Administrator receives new account request.	2) Potential User details (email address) are verified.
	3) User email is verified and considered authentic. User is added to Registered Users and Login details saved.
ALTERNATIVE ROUTE	
	3a) User email is wrong or not authentic. User is refused permission to register.

Non Functional Requirements:

- Security: System must ensure that User Login details are encrypted.

Validate new Album

ACTOR'S ACTION	SYSTEM RESPONSE
1) Administrator receives New Album request from a User.	2) User permissions are verified. User is prompted with Create new Album form.
ALTERNATIVE ROUTE	
	2a) User does not have permission to create a new album. System sends Refuse message. Event is logged

Monitor Activity

ACTOR'S ACTION	SYSTEM RESPONSE
1) Administrator examines through logs for anomalous activity.	2) Log reports are systematically examined through appropriate software.

Generate Report

ACTOR'S ACTION	SYSTEM RESPONSE
1) Administrator generates a report from monitored activity.	2) Report is generated, anomalous activities and failures are outlined.
3) Administrator generates bug reports if deemed necessary.	

Approve Campaign

ACTOR'S ACTION	SYSTEM RESPONSE
1) Administrator receives advert Campaign request from sponsor.	2) Sponsor credentials are verified. Campaign is examined for inappropriate content.
3) Administrator approves of campaign and agrees of terms and conditions.	4) Advertisement is added to the site.
ALTERNATIVE ROUTE	
	2a) Sponsor credentials are not verified or Campaign content is deemed inappropriate.
3a) Administrator contacts the sponsor to discuss any issue related to the campaign.	

Non Functional Requirements:

- Security: System must ensure that Sponsor and administrator communications and information exchange are secure.

Submit Campaign

ACTOR'S ACTION	SYSTEM RESPONSE
1) User contacts administrator for submitting a Campaign for the site.	2) A report is generated for later inspection by the administrator.
3) Administrator inspects the report and establishes if there are any issues to be discussed with the Sponsor.	4) Sponsor is notified that Campaign will be added to the site.
ALTERNATIVE ROUTE	
3a) Administrator finds issues with the Advertisement Campaign.	4a) Sponsor is notified of issues in removing the campaign. Further communication exchange between sponsor and administrator needed.

Non Functional Requirements:

- Security: System must ensure that Sponsor and administrator communications and information exchange are secure.

Remove campaign

ACTOR'S ACTION	SYSTEM RESPONSE
1) User contacts administrator for removing a Campaign from the site.	2) A report is generated for later inspection by the administrator.
3) Administrator inspects the report and establishes if there are any issues to be discussed with the Sponsor	4) Sponsor is notified that Campaign will be removed.
ALTERNATIVE ROUTE	
3a) administrator finds issues to be discussed with the sponsor.	4a) Sponsor is notified of issues in removing the campaign. Further communication exchange between sponsor and administrator needed.

Non Functional Requirements:

- Security: System must ensure that Sponsor and administrator communications and information exchange are secure.

Detailed Use Case Description

USE CASE: CREATE ALBUM		
DESCRIPTION:		Use case defines the process and interactions between end User and system when the option "Create new Album" is selected by the User.
POSTCONDITIONS:		User is logged in.
SUCCESS END CONDITION:		Album is created properly.
FAILED END CONDITION:		Failed to create a new Album
ACTORS:		User, System.
TRIGGER:		"Create new Album" is selected by the User from the site GUI.
FLOW		
DESCRIPTION	**STEP NUMBER**	**ACTION**
	1	"Create new Album" procedure called by the user through site GUI.
	2	User Permissions and privileges are verified by the System.
	3	User is prompted with Create new Album form to fill in..
	4	User fills form and selects confirm.
	5	For is verified by the system for correctness.
	6	Album is created and added to user's album list.
EXTENSION	**STEP NUMBER**	**BRANCHING ACTION**
VARIATION	**STEP NUMBER**	**BRANCHING ACTION**
		2a) User does not have permissions to create a new album.
		4a) User fills Create new Album Form wrongly.

Tactics Adopted for Quality Attributes

A number of Quality attributes are supported through architectural decisions. These quality attributes include:

- Portability: To aid portability two large architectural decisions have been made, namely: Java will be the primary language and potential usage of an application server such as Glassfish at a later iteration in design can be considered also. Java is a language capable of running on a large number of platforms; its use would drastically simplify deploying the software to users across a wide range of computer environments. The use of Glassfish will help portability through its ability to quickly and easily create a web interface. It provides middleware to deploy services to a large number of people through a web browser.

- Reliability: Reliability will be enforced through the use of replication; it is likely that RAID 6 will be used for this purpose. It is of utmost importance that media uploaded by the user remains accessible at all times. RAID 6 provides the level of reliability required, being capable of withstanding multiple simultaneous drive failures. As mentioned above, use of Glassfish would also support reliability. Glassfish implements services like fail-over, ensuring the system can handle abnormal termination of an active server.

- Extensibility: Extensibility is provided through the use of sound programming practices. Most notably one should follow the principle "program to interfaces not to implementation". Through the use of design patterns it will be evident that the base code can be extended with minimal modification.

GUI/layout Screenshots

Login form

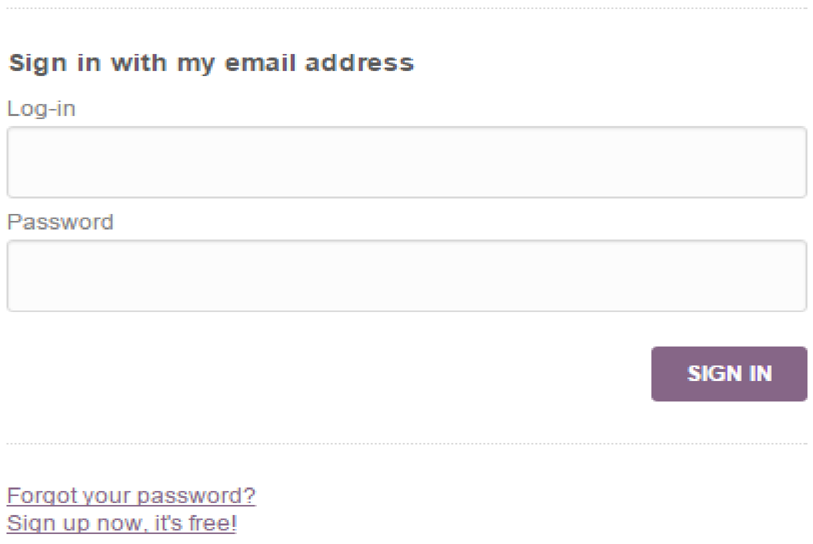


Illustration 2

Create new Account Form

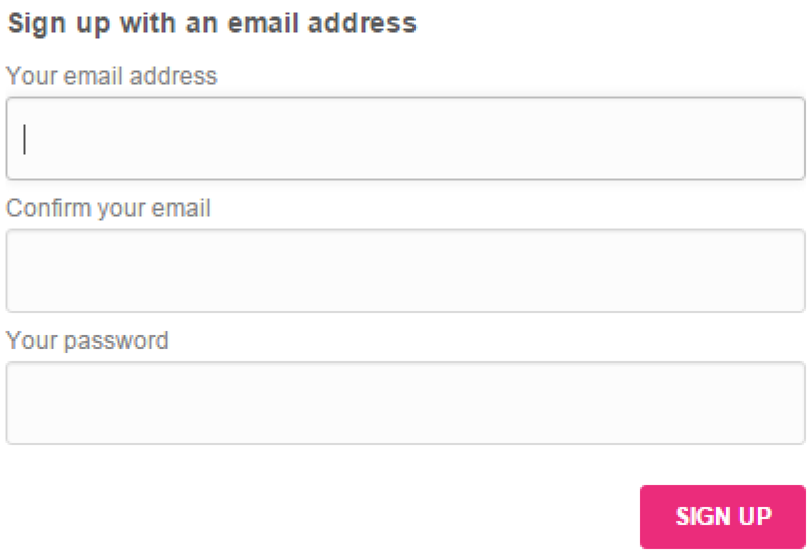
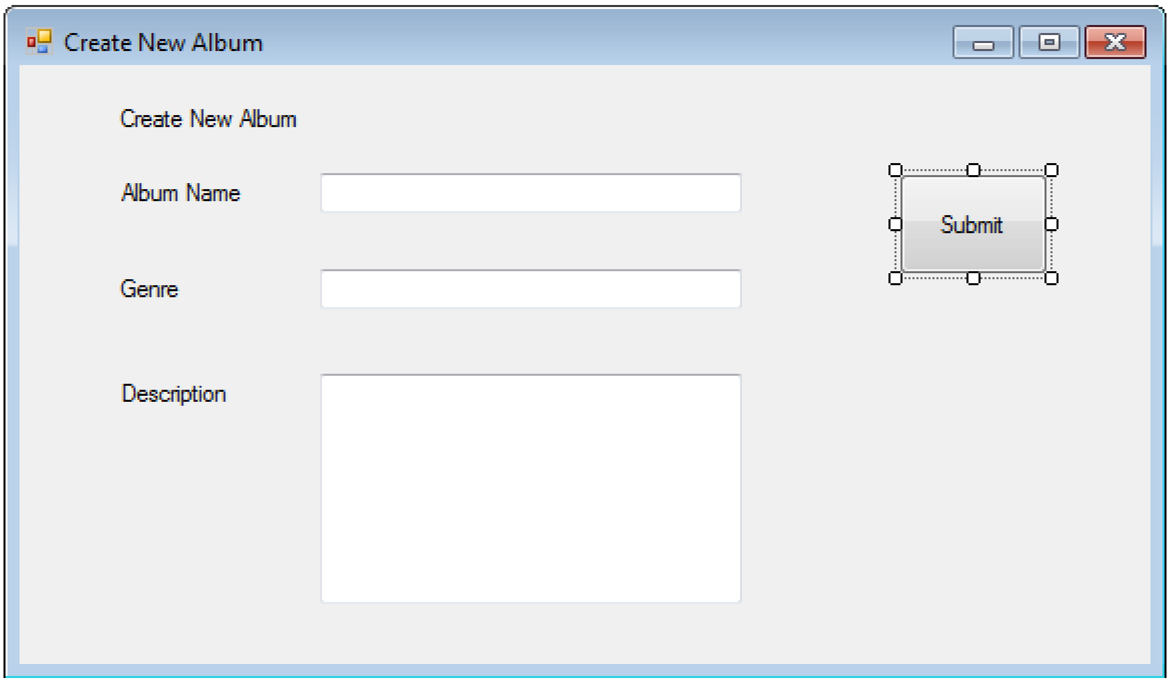
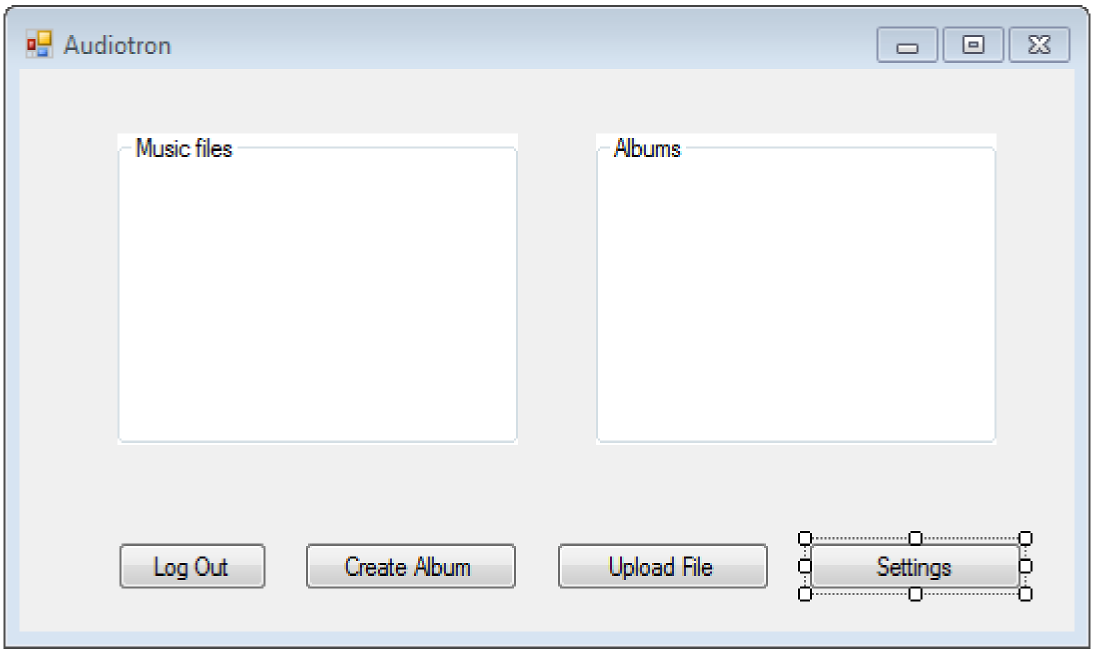


Illustration 3

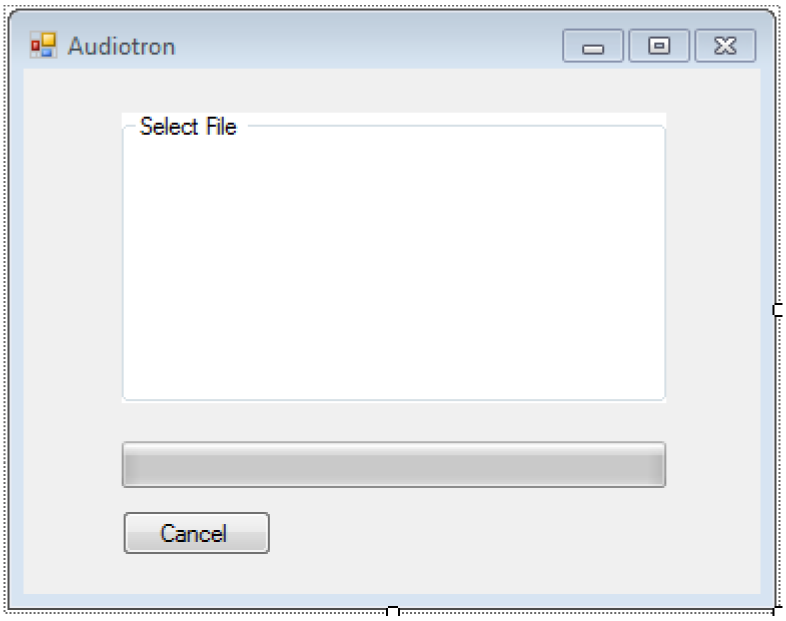
Create Album form



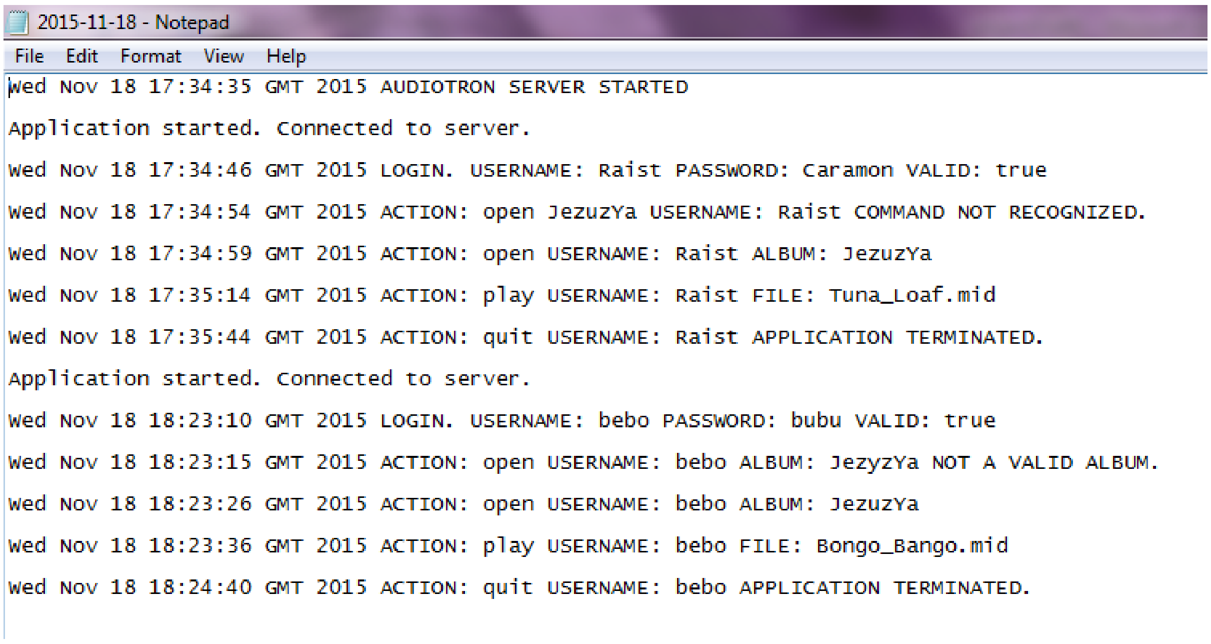
Main Activity



Upload /Download file



Logs/Reports



```
Wed Nov 18 17:34:35 GMT 2015 AUDIOTRON SERVER STARTED
Application started. Connected to server.
Wed Nov 18 17:34:46 GMT 2015 LOGIN. USERNAME: Raist PASSWORD: Caramon VALID: true
Wed Nov 18 17:34:54 GMT 2015 ACTION: open JezuzYa USERNAME: Raist COMMAND NOT RECOGNIZED.
Wed Nov 18 17:34:59 GMT 2015 ACTION: open USERNAME: Raist ALBUM: JezuzYa
Wed Nov 18 17:35:14 GMT 2015 ACTION: play USERNAME: Raist FILE: Tuna_Loaf.mid
Wed Nov 18 17:35:44 GMT 2015 ACTION: quit USERNAME: Raist APPLICATION TERMINATED.
Application started. Connected to server.
Wed Nov 18 18:23:10 GMT 2015 LOGIN. USERNAME: bebo PASSWORD: bubu VALID: true
Wed Nov 18 18:23:15 GMT 2015 ACTION: open USERNAME: bebo ALBUM: JezyzYa NOT A VALID ALBUM.
Wed Nov 18 18:23:26 GMT 2015 ACTION: open USERNAME: bebo ALBUM: JezuzYa
Wed Nov 18 18:23:36 GMT 2015 ACTION: play USERNAME: bebo FILE: Bongo_Bango.mid
Wed Nov 18 18:24:40 GMT 2015 ACTION: quit USERNAME: bebo APPLICATION TERMINATED.
```

System Architecture

Architectural Decisions

The system can be logically divided into three subsystems:

- A Server System, capable of dealing with multiple queries from multiple users concurrently. The server does not actually resolve the queries, its function is mainly to queue requests according to timestamp based priorities or privileges. The server deals also with the download mechanics of files. Protocols and locking mechanisms have to be designed and implemented to coordinate users' activities. Activities must be monitored to prevent bottleneck or overflow of server resources.
- A database system that deals with the queries sent by the server. The database must be secure, fast and consisting of high volume storage units. The database tables should follow a predefined structure and relationships should be established between such tables to facilitate browsing materials related to each other by some common topic. User information and login details are also stored here.
- A Client system, through which each user can interact with the application interface. The user system is also responsible for the upload mechanics. Clients do not have direct access to the database

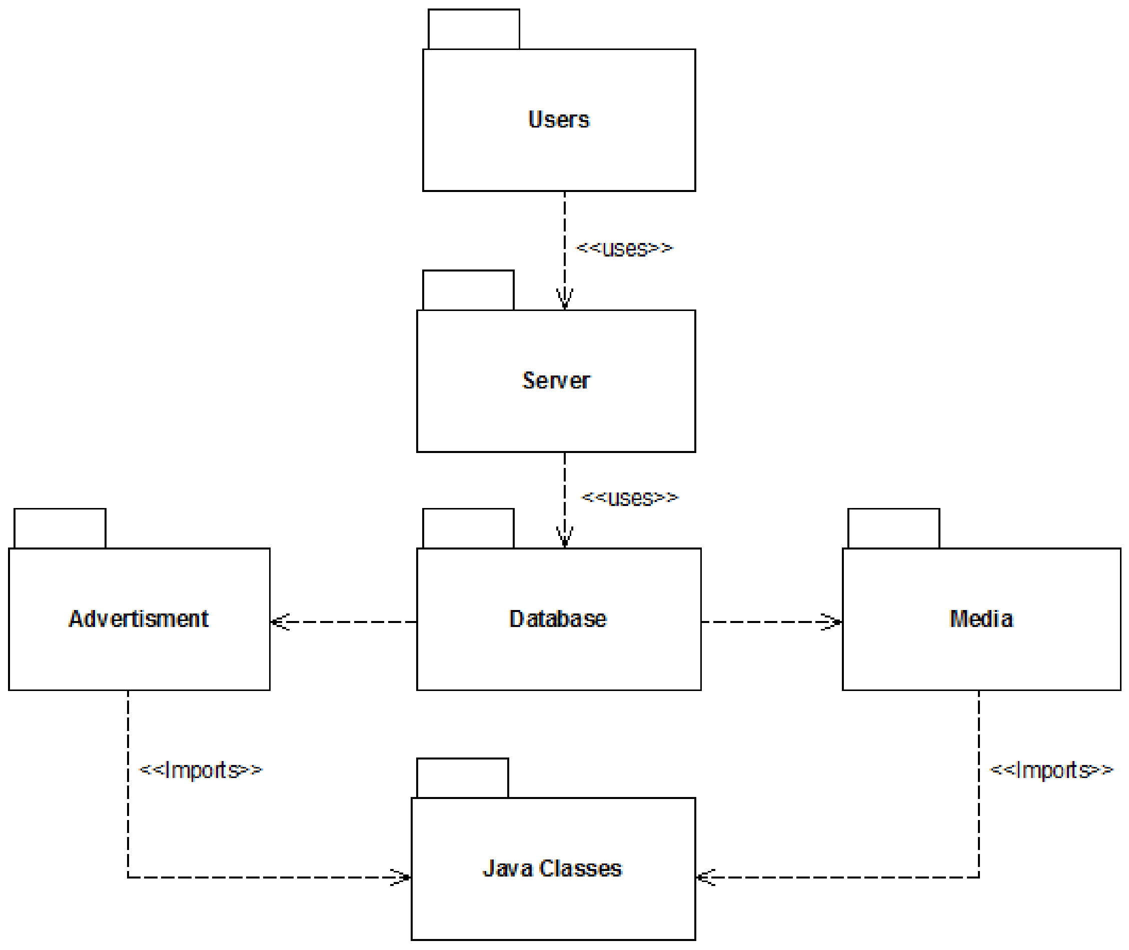
Implementation Language

The group has agreed on the superiority of C++ over other languages when it comes to performance; however a decision has been made to adopt Java for the initial implementation. Java is the perfect choice when considering portability over distributed systems. Moreover through Java extensive API one can easily implement databases and client server communication routines.

UML workbench

For its quick learning timeline, ease of use and high quality diagrams, the group has decided to use Cadifra UML Editor 1.3 by Adrian & Frank Buehlmann for the realization of most of the diagrams presented in this document. Gliffy.com has also been used. Gliffy.com offers a wider variety of features to supplement UML diagram design. The use of the Gliffy.com website also enabled project members to easily access the most up to date diagrams.

Package Diagram



Lightweight Analysis Artefacts

The group has decided upon a Data Driven Design approach to list any possible object to implement.

List of Candidate Objects

To find appropriate candidate objects, the noun identification technique has been applied to the described use cases. Simple heuristic has been used to identify potential objects.

From User Use Cases:

User	Form	Username	Password	Login
Register	Email	Guest	Main Activity	Media
Error (Bug)	Album	Inspection	Permissions	Computer
Server	Download	Rank	Comment	(MediaPlayer)

From Administrator\Buisness Tier Use Cases:

Administrator	Software	Reports	Campaign	Sponsor
Advertisment				

<u>Heuristics</u>

() = Inferred Objects

Green : Potential Class

Yellow : Attribute

Red : Too Specific or Vague

Blue : Event

Purple : Out Of Scope

Based on the noun identification technique, the group has decided upon 7 potential candidate classes:

1. User

2. Media

3. Album

4. Comment

5. Reports

6. Campaign

7. Advertisement

8. Server

User

User is a generic to identify the person using the system (i.e.: Administrator, Guest, etc.). The user class will be responsible for determining actor's permissions, the functions that available to him/her. Further information can be stored there such as preferences, friend list etc.

Media

This entity specifies a media file that can be played by the system. A Media has an owner that allows users to listen to it. It can be added as part of an album.

Album

A collection of Media objects that can be updated or deleted by the user that owns it.

Comment

A Comment consists of a username, a rank (one to five stars) and a message. Comments can be used to give artists feedback about their music as well as a way for users to express their opinions about available albums and playlists.

Report

A Report logs any event, change, request or similar action performed on the system that requires Administrator's approval or monitoring.

Campaign

This entity represents a collection of Advertisements proposed by the same sponsor.

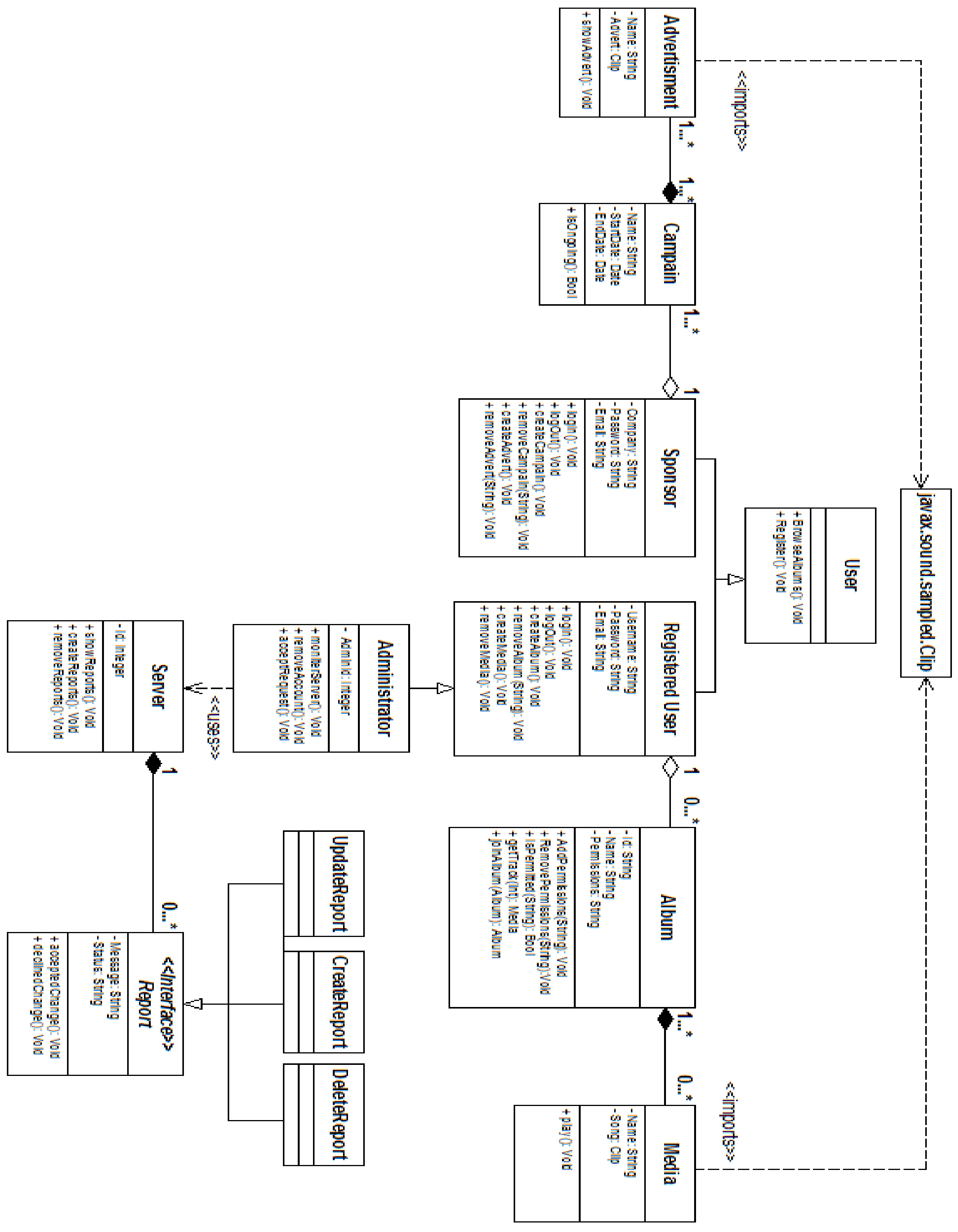
Advertisement

An advertisement is a media file (image, audio, video, etc.) proposed by a sponsor as part of a Campaign. Advertisements are shown to any user browsing the site. The site administrator can decide whether an advertisement is appropriate for display or not.

Server

The server is the entity responsible for the coordination between clients and the system database. It also validates permissions and creates reports.

Class Diagram



Sequence Diagram

Log in

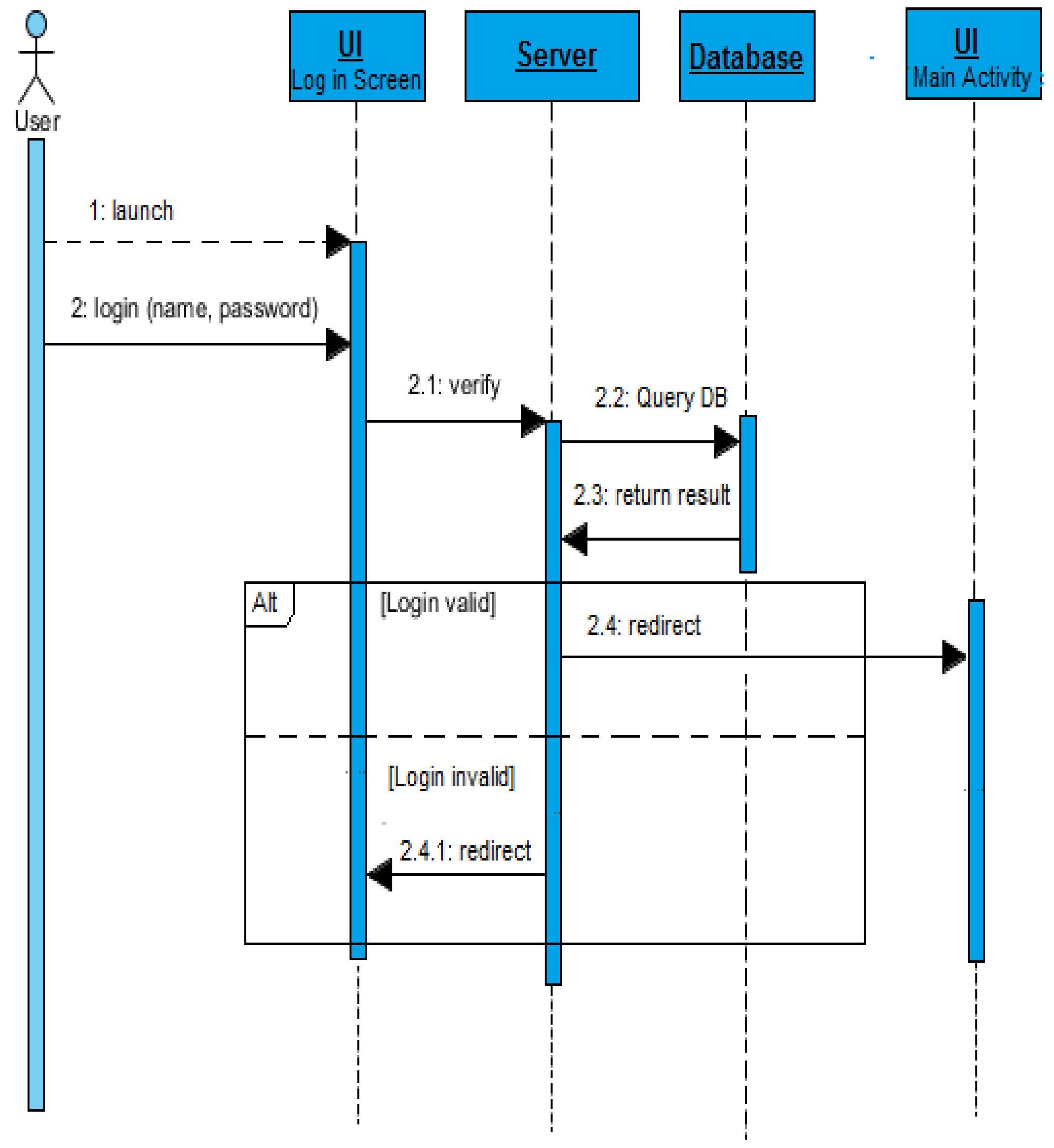


Illustration 4

Main Activity

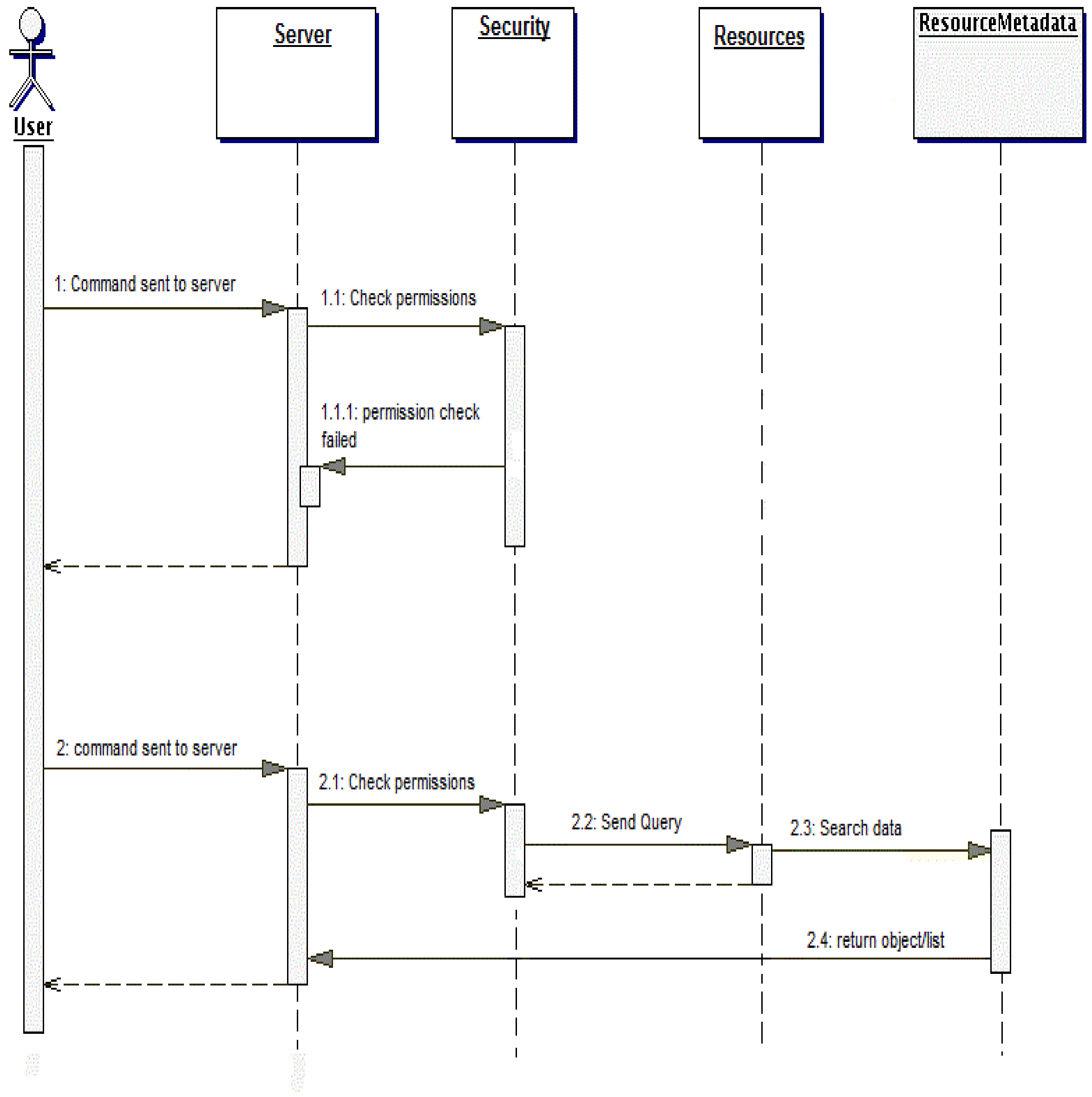
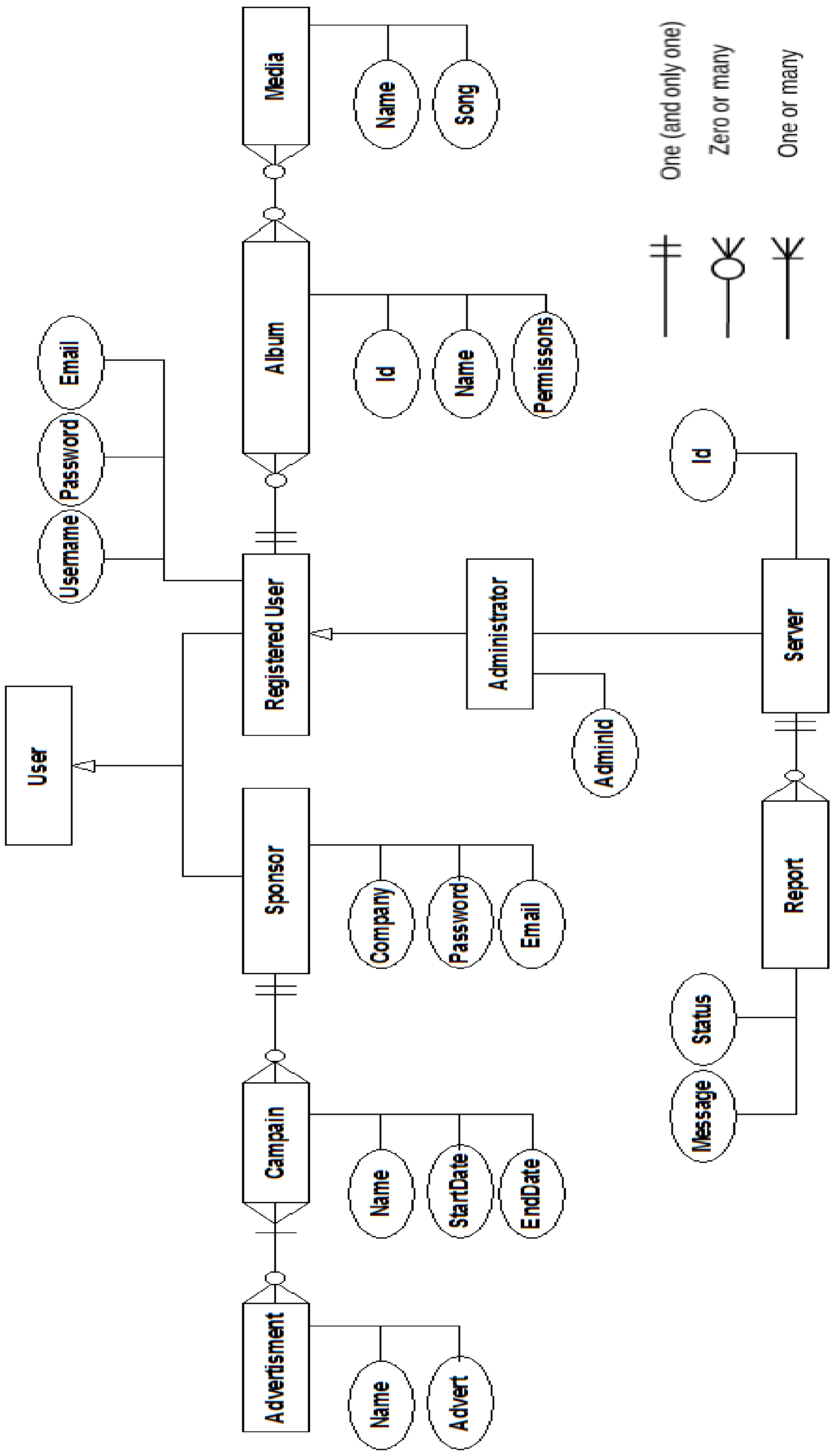


Illustration 5

Entity Relationship Diagram



Code

Package: audiotronServer

User.java

```
package audiotronserver;

public interface User
{
	public boolean IsAdmin();
	public boolean IsRegistered();
}
```

Administrator.java

```
package audiotronserver;

public class Administrator extends RegisteredUser
{
	private int adminId;

	public Administrator (String newUser, String newPass,String newEmail,int adminNumber)
	{
		super(newUser,newPass,newEmail);
		adminId = adminNumber;
	}

	public int getId()
	{
		return adminId;
	}

	public boolean IsRegistered()
	{
		return true;
	}

	public boolean IsAdmin()
	{
		return true;
	}
}
```

RegisteredUser.java

```
package audiotronserver;
```

```
public class RegisteredUser implements User {

	private String username;
	private String password;
	private String email;

	public RegisteredUser (String newUser, String newPass,String newEmail)
	{
		username = newUser;
		password = newPass;
		email = newEmail;
	}

	public String getName()
	{
		return username;
	}

	public String getPass()
	{
		return password;
	}

	public String getEmail()
	{
		return email;
	}

	public boolean IsRegistered()
	{
		return true;
	}

	public boolean IsAdmin()
	{
		return false;
	}
}
```

Report.java

```
package audiotronserver;

public class Report
{
	private String reportText;

	public Report(){
		reportText = "-------------General Server Information-------------\n\n";
		reportText += "Server running time : ";
		reportText += InformationManager.getInstance().getTimeElapsed() + "\n";
		reportText += "--------------------------------------------------\n";
	}

	public void printReport(){
		System.out.println(reportText);
	}
}
```

ReportDecorator.java

```
package audiotronserver;

abstract public class ReportDecorator extends Report{
	private Report myReport;

	public ReportDecorator(Report newReport){
		myReport = newReport;
	}

	public void callPrintReport(){
		if(myReport != null)
			myReport.printReport();
	}
}
```

UserReport.java

```
package audiotronserver;

public class UserReport extends ReportDecorator{

	private String userName;
	private String reportString;

	public UserReport(Report newReport, String userName){
		super(newReport);
		this.userName = userName;
		reportString = "-------------User Information-------------\n";
		reportString += InformationManager.getInstance().getUserRecord(userName) +
"\n";
		reportString += "----------------------------------------\n";
	}

	public void printReport(){
		super.callPrintReport();
		System.out.println(reportString);
	}
}
```

DetailsVerifyer.java

```
package audiotronserver;

import java.io.*;
import java.text.*;
import java.util.*;

public class DetailsVerifyer
{
	public static boolean validUser(String username, String password)
	{
		try{
		File file = new File("Account.txt");
			if (file.exists())
			{
				Scanner intake = new Scanner(file);
				while (intake.hasNext())
				{
					String [] userDetailArray = intake.next().split(",");
					//System.out.println(userDetailArray.length);
					if(userDetailArray.length == 3)
					{
					//System.out.println(userDetailArray[0] + " " +
userDetailArray[1] + " " + userDetailArray[2]);
						if(userDetailArray[0].equalsIgnoreCase(username)
&& userDetailArray[1].equalsIgnoreCase(password)){
							intake.close();
							return true;
						}
					}
				}
				intake.close();
				return false;
			}
		}
		catch (Exception ex){

		}
		return false;

	}
}
```

InformationManager.java

```
package audiotronserver;

import java.io.*;
import java.util.*;

public class InformationManager
{

	private Scanner in;
	private static InformationManager informationPlayerInstance;
	private long startTime;
	private ArrayList<String> userRecords;

	private InformationManager()
	{
		userRecords = new ArrayList<String>();
	}

	public static InformationManager getInstance()
	{
		if(informationPlayerInstance == null)
			informationPlayerInstance = new InformationManager();
		return informationPlayerInstance;
	}

	public void startTimeLogging(){
		startTime = System.currentTimeMillis();
	}

	public long getTimeElapsed(){
		return System.currentTimeMillis() - startTime;
	}

	public void addUserRecord(String userName, String record){
		int index = findUser(userName);
		if(index == -1){
			//System.out.println("creating record for " + userName);
			String newRecord = userName + " : \n" + record + "\n";
			userRecords.add(newRecord);
			//System.out.println(newRecord);
		}
		else{
			userRecords.set(index, userRecords.get(index) + record + "\n");
		}
	}

	public String getUserRecord(String userName){
		int index = findUser(userName);
		if(index == -1){
			return "Error : no data on user";
		}
		else{
			return userRecords.get(index);
		}
	}

	private int findUser(String userName){
		//System.out.println("Trying to find " + userName);
		for(int i = 0; i < userRecords.size(); i++){
			int firstWordSplit = userRecords.get(i).indexOf(' ');
			String word = userRecords.get(i).substring(0, firstWordSplit);
			//System.out.println("index " + i + " word " + word);
			if(userName.equals(word)){
				return i;
			}
		}
		return -1;
	}
}
```

AudiotronServer.java

```
package audiotronserver;

import java.io.*;
import java.text.*;
import java.util.*;
import media.MediaPlayer;
import media.MediaFile;
import media.MediaFileCreator;

public class AudiotronServer
{
	private MediaFile currentMediaInstance = null;
	private ArrayList<MediaFile> initialMediaList = new ArrayList<MediaFile>();
	private ArrayList<MediaFile> advertisementList = new ArrayList<MediaFile>();
	private String userName = "";
	private MediaFileCreator mediaFileCreator;

	public AudiotronServer()
	{
		//System.out.println("Creating server");
		InformationManager.getInstance().startTimeLogging();
		mediaFileCreator = new MediaFileCreator();
		instansiateArrays();
		instansiateAdvertisments();
		for(int i = 0; i < initialMediaList.size(); i++){
			//System.out.println((initialMediaList.get(i)).getName());
		}
	}

	public boolean validUser(String name, String password)
	{
		//System.out.println(name + " " + password);
		if(DetailsVerifyer.validUser(name, password)){
			userName = name;
			return true;
		}
		else{
			return false;
		}

	}

	public boolean validAlbum(String album){
		InformationManager.getInstance().addUserRecord(userName, "Trying to access " +
album);
		if(currentMediaInstance == null){
			for(int i = 0; i < initialMediaList.size(); i++){
				if(album.equals(initialMediaList.get(i).getName())){
					currentMediaInstance = initialMediaList.get(i);
					return true;
				}
			}
			return false;
		}
		ArrayList<MediaFile> tempMediaFileList = currentMediaInstance.getMediaList();
		for(int i = 0; i < tempMediaFileList.size(); i++){
				if(album.equals(tempMediaFileList.get(i).getName())){
					currentMediaInstance = tempMediaFileList.get(i);
					return true;
				}
			}
		return false;

	}
	public void returnFromAlbum(){
		currentMediaInstance = currentMediaInstance.getParent();

	}

	public boolean validFileToPlay(String fileString){
		InformationManager.getInstance().addUserRecord(userName, "Trying to play " +
fileString);
		if(currentMediaInstance == null){
			for(int i = 0; i < initialMediaList.size(); i++){
				if(fileString.equals(initialMediaList.get(i).getName())){
					//chance to play an add
					if(Math.random() < 0.25){
```

```
                                        MediaFile newAd = getRandomAdd();

            InformationManager.getInstance().addUserRecord(userName, "Playing ad " +
newAd.getName());
                                        MediaPlayer.getInstance().playAd(newAd);
                                        MediaPlayer.getInstance().reset();
                                    }

                                    initialMediaList.get(i).play();
                                    MediaPlayer.getInstance().reset();
                                    return true;
                                }
                            }
                            return false;
                    }
                    ArrayList<MediaFile> tempMediaFileList = currentMediaInstance.getMediaList();
                    for(int i = 0; i < tempMediaFileList.size(); i++){
                                if(fileString.equals(tempMediaFileList.get(i).getName())){
                                    tempMediaFileList.get(i).play();
                                    MediaPlayer.getInstance().reset();
                                    return true;
                                }
                            }
                    return false;

            }

            public ArrayList<String> getCurrentMediaList(){
                    ArrayList<String> mediaList = new ArrayList<String>();
                    if(currentMediaInstance == null){
                            for(int i = 0; i < initialMediaList.size(); i++){
                                mediaList.add("Album : " + (initialMediaList.get(i)).getName());

                            }
                            return mediaList;
                    }
                    ArrayList<MediaFile> tempMediaFileList = currentMediaInstance.getMediaList();
                    for(int i = 0; i < tempMediaFileList.size(); i++){
                                if(tempMediaFileList.get(i).isAlbum())
                                    mediaList.add("Album : " +
tempMediaFileList.get(i).getName());
                                else
                                    mediaList.add("Audio : " +
tempMediaFileList.get(i).getName());
                            }
                    return mediaList;
            }

            private void instansiateAdvertisments()
            {
                    try{
```

```
		File file = new File("Advertisements.txt");
			if (file.exists())
			{
				Scanner intake = new Scanner(file);
				ArrayList<MediaFile> previousAlbums = new ArrayList<MediaFile>();
				//System.out.println("File exists, scanner created");
				while (intake.hasNext())
				{
					String adName = intake.next();
					MediaFile newFile =
mediaFileCreator.createMediaFile(adName, "audio");
					advertisementList.add(newFile);
				}
			}
		}
		catch(Exception e){

		}
	}

	private MediaFile getRandomAdd(){
		int adIndex = (int) Math.random()*advertisementList.size();
		return advertisementList.get(adIndex);
	}

	private void instansiateArrays()
	{
		//System.out.println("Instansiating Arrays");
		try{
		File file = new File("AlbumInfo.txt");
			if (file.exists())
			{
				Scanner intake = new Scanner(file);
				ArrayList<MediaFile> previousAlbums = new ArrayList<MediaFile>();
				//System.out.println("File exists, scanner created");
				while (intake.hasNext())
				{
					String albumArray[] = intake.nextLine().trim().split(" ");
					if(albumArray.length != 0){
						if(albumArray[0].equals("Album")){
							//System.out.println("Creating new Album
of name : " + albumArray[1]);
							MediaFile newFile =
mediaFileCreator.createMediaFile(albumArray[1], "album");
							previousAlbums.add(newFile);
							//System.out.println("Adding to
previousAlbums, new size : " + previousAlbums.size());
						}
						else if(albumArray[0].equals("}")){
							if(previousAlbums.size() != 1){
```

```
                                                                //System.out.println("Destroying
child album link and adding to parent, album name : " + previousAlbums.get(previousAlbums.size()-
1).getName());

        previousAlbums.get(previousAlbums.size()-
2).addMedia(previousAlbums.get(previousAlbums.size()-1));

        previousAlbums.get(previousAlbums.size()-
1).setParent(previousAlbums.get(previousAlbums.size()-2));

        previousAlbums.remove(previousAlbums.size()-1);
                                                                //System.out.println("Removing
from previousAlbums, new size : " + previousAlbums.size());
                                                        }
                                                        else{
                                                                //System.out.println("Destroying
root album link and adding to mediaList, album name : " +
previousAlbums.get(previousAlbums.size()-1).getName());

        initialMediaList.add(previousAlbums.get(previousAlbums.size()-1));

        previousAlbums.remove(previousAlbums.size()-1);
                                                        }
                                                }
                                                else if(albumArray[0].equals("{")){
                                                        //System.out.println("starting new array
filling process");
                                                }

                                                else{
                                                        //System.out.println("Adding media : " +
albumArray[0] + " to : " + previousAlbums.get(previousAlbums.size()-1).getName());
                                                        MediaFile newFile =
mediaFileCreator.createMediaFile(albumArray[0], "audio");
                                                        //System.out.println("Media name
confirmation : " + newFile.getName());
                                                        (previousAlbums.get(previousAlbums.size()-
1)).addMedia(newFile);

                                                }
                                        }
                                }
                                intake.close();
                        }
                }
                catch (Exception ex){}
        }

        public Report generateReport(){
                return new Report();
        }
```

```
	public Report generateReport(String userName){
		return new UserReport(new Report(), userName);
	}

}
```

Package: media

MediaFile.java

```
package media;

import java.util.*;

public abstract class MediaFile
{
	private String name;

	public MediaFile(String name)
	{
		this.name = name;
	}

	//Extended by album class
	public Album open()
	{
		return null;
	}

	public void addMedia(MediaFile media)
	{
	}

	public void remove(MediaFile media)
	{
	}

	public void addPermission(String user){
	}

	public String getName()
	{
		return name;
	}
	public MediaFile getParent()
	{
		return null;
	}
	public void setParent(MediaFile newFile)
	{
	}

	public boolean isAlbum()
	{
		return false;
	}

	public ArrayList<MediaFile> getMediaList()
	{
		return null;
	}

	public abstract void play();
}
```

Audio.java

```
package media;

import javax.sound.sampled.*;
import java.io.*;

public class Audio extends MediaFile
{
	Audio (String fileName)
	{
		super(fileName);
	}


	public void play()
	{
		MediaPlayer.getInstance().play(this);
	}
}
```

MediaFileFactory.java

```
package media;

public interface MediaFileFactory{

	public MediaFile createMediaFile(String name, String type);

}
```

MediaFileCreator.java

```
package media;

public class MediaFileCreator implements MediaFileFactory{

	public MediaFile createMediaFile(String name, String type){
		if(type.equals("audio")){
			return new Audio(name);
		}
		else if(type.equals("album")){
			return new Album(name);
		}
		return null;
	}
}
```

Album.java

```
package media;

import java.text.*;
import java.util.*;

public class Album extends MediaFile
{
	private ArrayList<MediaFile> mediaList = new ArrayList<MediaFile>();
	private MediaFile parent;

	public Album (String albumName)
	{
		super(albumName);
		this.parent = null;
	}
	public void setParent(MediaFile media)
	{
		this.parent = media;
	}

	public void addMedia(MediaFile media)
	{
		mediaList.add(media);
	}

	public void addPermission(String user)
	{
	}

	public void play()
	{
		for(int i = 0; i < mediaList.size(); i++)
			mediaList.get(i).play();
	}

	public MediaFile getParent()
	{
		return parent;
	}

	public ArrayList<MediaFile> getMediaList()
	{
		return mediaList;
	}

	public boolean isAlbum()
	{
		return true;
	}

}
```

MediaPlayer.java

```
package media;

import javax.sound.sampled.*;
import java.io.*;
import java.util.Scanner;

public class MediaPlayer
{
	private static MediaPlayer mediaPlayerInstance;
	private boolean stopPlaying = false;
	private Scanner in;

	private MediaPlayer()
	{
	}

	public static MediaPlayer getInstance()
	{
		if(mediaPlayerInstance == null)
			mediaPlayerInstance = new MediaPlayer();
		return mediaPlayerInstance;
	}

	public void play(Audio audioInstance)
	{
		if(stopPlaying == false){
```

```
                        try
                                {
                                        in = new Scanner(System.in);
                                        File musicFile = new File(audioInstance.getName());
                                        System.out.println("Loading " + audioInstance.getName());
                                        AudioInputStream audioIn = 
AudioSystem.getAudioInputStream(musicFile);
                                        Clip clip = AudioSystem.getClip();
                                        clip.open(audioIn);
                                        clip.start();
                                        System.out.println("Playing " + audioInstance.getName());
                                        boolean invalidCommand = true;
                                        while(invalidCommand == true){
                                                System.out.print("Enter 'stop' to stop playing, or 
'finish' to clear current play list : ");
                                                String input = in.nextLine().trim();
                                                if(input.equals("stop")){
                                                        clip.stop();
                                                        invalidCommand = false;
                                                }
                                                else if(input.equals("finish")){
                                                        stopPlaying = true;
                                                        clip.stop();
                                                        invalidCommand = false;
                                                }
                                                else
                                                        System.out.println("command not 
recognized");
                                        }
                                }
                                catch (Exception ex){System.out.println(ex.toString());}
                }
        }

        public void playAd(MediaFile audioInstance)
        {
                if(stopPlaying == false){
                        try
                                {
                                        in = new Scanner(System.in);
                                        File musicFile = new File(audioInstance.getName());
                                        System.out.println("Loading ad" + 
audioInstance.getName());
                                        AudioInputStream audioIn = 
AudioSystem.getAudioInputStream(musicFile);
                                        Clip clip = AudioSystem.getClip();
                                        clip.open(audioIn);
                                        clip.start();
                                        System.out.println("Playing ad" + audioInstance.getName());
                                        while(clip.isActive()){
                                        }
                                }
                        catch (Exception ex){System.out.println(ex.toString());}
                }
        }

        public void reset(){
                stopPlaying = false;
        }
}
```

Package: userInterface

MainView.java

```
package userInterface;

import java.util.*;
import java.text.*;
import java.io.*;
import audiotronserver.AudiotronServer;

public class MainView
{
        //log in, check with server, if accepted ->
        //retrieve list of mediafiles
        //send commands to server to utilize any of these mediaFiles.

        private static Scanner in;
        private static AudiotronServer ATServer;

        public static void main(String [] args)
        {
                in = new Scanner(System.in);
                ATServer = new AudiotronServer();
                boolean viewing = true;
                while(viewing == true)
                {
                        if(logIn())
                        {
                                //code here for browsing and playing
                                boolean albumView = true;
                                while(albumView){
                                        boolean invalidCommand = true;
                                        System.out.println("You have access to the following files");
                                        ArrayList<String> albumList =
ATServer.getCurrentMediaList();
                                        for(int i = 0; i < albumList.size(); i++){
                                                System.out.println("\t" + albumList.get(i));
                                        }
```

```
                                                while(invalidCommand){
                                                System.out.print("Enter command : ");
                                                String action = in.nextLine().trim();
                                                        switch(action){
                                                                case "open": {
                                                                        System.out.print("\nEnter album to
open : ");
                                                                        String albumString =
in.nextLine().trim();

        if(!ATServer.validAlbum(albumString)){
                                                                                System.out.println("Not a
valid album");
                                                                        }
                                                                        invalidCommand = false;
                                                                        break;
                                                                }

                                                                case "return": {
                                                                        ATServer.returnFromAlbum();
                                                                        invalidCommand = false;
                                                                        break;
                                                                }

                                                                case "quit": {
                                                                        System.exit(0);
                                                                        invalidCommand = false;
                                                                        break;
                                                                }

                                                                case "play": {
                                                                        System.out.print("\nEnter file to
play : ");
                                                                        String fileString =
in.nextLine().trim();

        if(!ATServer.validFileToPlay(fileString)){
                                                                                System.out.println("Not a
valid file");
                                                                        }
                                                                        invalidCommand = false;
                                                                        break;
                                                                }

                                                                case "generate report": {
                                                                        //System.out.print("\nEnter file to
play : ");
                                                                        //String fileString =
in.nextLine().trim();

        //if(!ATServer.validFileToPlay(fileString)){
										//	System.out.println("Not a
valid file");
										//}

	ATServer.generateReport().printReport();
										invalidCommand = false;
										break;
									}

									case "generate user report": {
										System.out.print("\nEnter user to
generate for : ");
										String userString =
in.nextLine().trim();

	//if(!ATServer.validFileToPlay(fileString)){
										//	System.out.println("Not a
valid file");
										//}

	ATServer.generateReport(userString).printReport();
										invalidCommand = false;
										break;
									}

									default:
										System.out.println("command not
recognized");
										break;



								}
							}
						}
					}
					System.out.println("Do you wish to terminate program\nY/N: ");
					String terminateString = in.nextLine();
					if(terminateString.equals("Y") || terminateString.equals("y"))
						viewing = false;
				}

	}

	private static boolean logIn()
	{
		System.out.print("Please Enter login details\nName: ");
		String name = in.nextLine();
		System.out.print("Password: ");
		String password = in.nextLine();
		if(ATServer.validUser(name, password)){
			System.out.println("Log in details accepted");
			return true;
		}
		else{
			System.out.println("Log in details invalid");
			return false;
		}

	}

	private static void printCurrentMediaList()
	{
		ATServer.getCurrentMediaList();
	}

}
```

Design Artefacts

Adopted Design Patterns

The group has agreed upon applying three design patterns during code implementation:

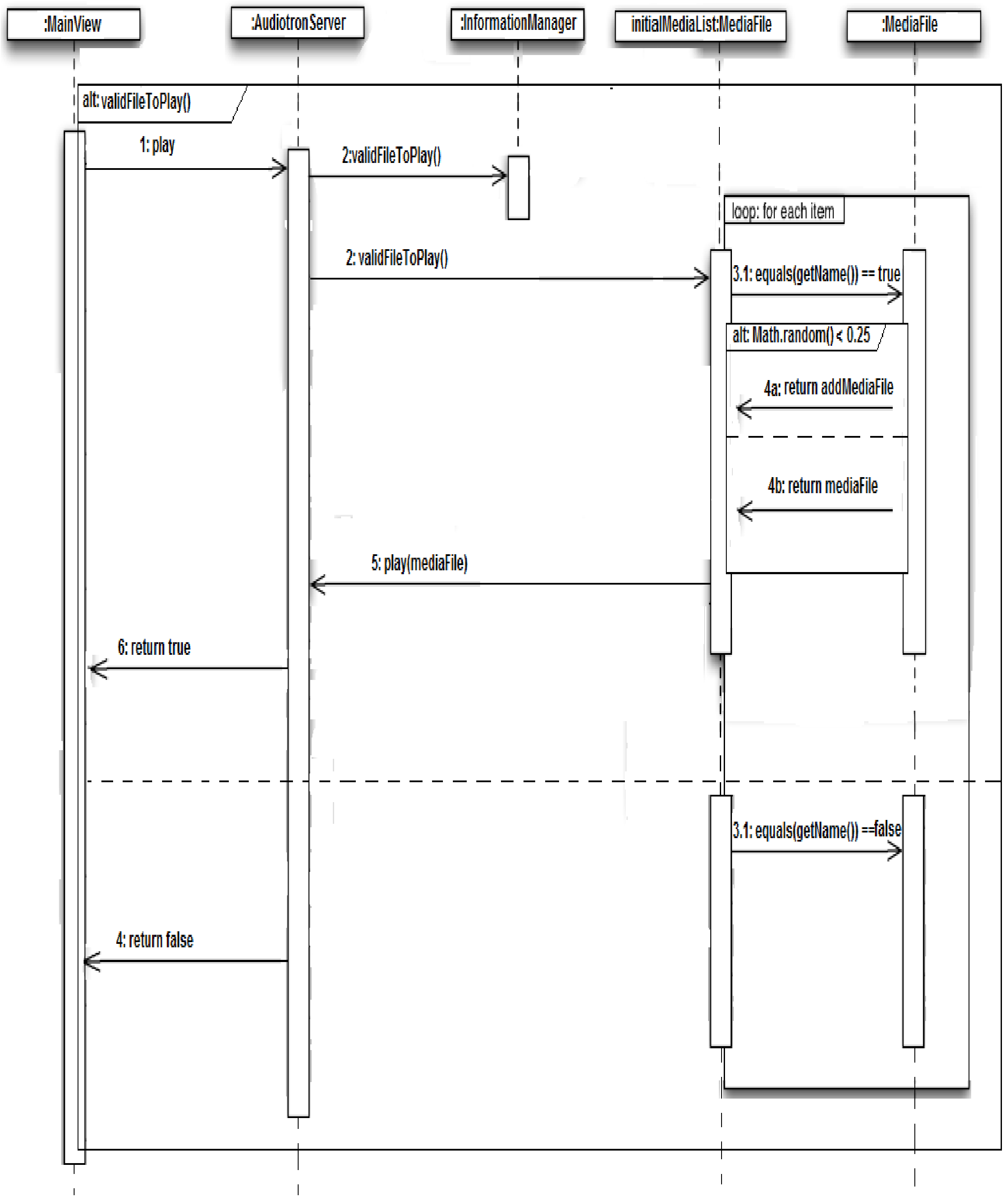
1) **Decorator pattern**: It is possible for system administrators to generate and print reports concerning users and database activities. An administartor may want to create a report with various different features. The decorator design pattern seemed the most suitable solution to prevent a combinatorial explosion of subclasses, (i.e.: SystemAndUserReport, TransactionAndUserReport etc…). Reports can be created with different features. As a consequence for applying such pattern, the list of features available for reports can be easily extended or updated.

2) **Factory Method**: The factory method design pattern was chosen for creating objects of type MediaFile. As the server creates a large volume of these objects, the system needs to maintain a high cohesion. By applying the factory method design pattern one also supports extensibility (i.e.: a change to album class' constructor only has to be accounted for in the factory class and not the server).

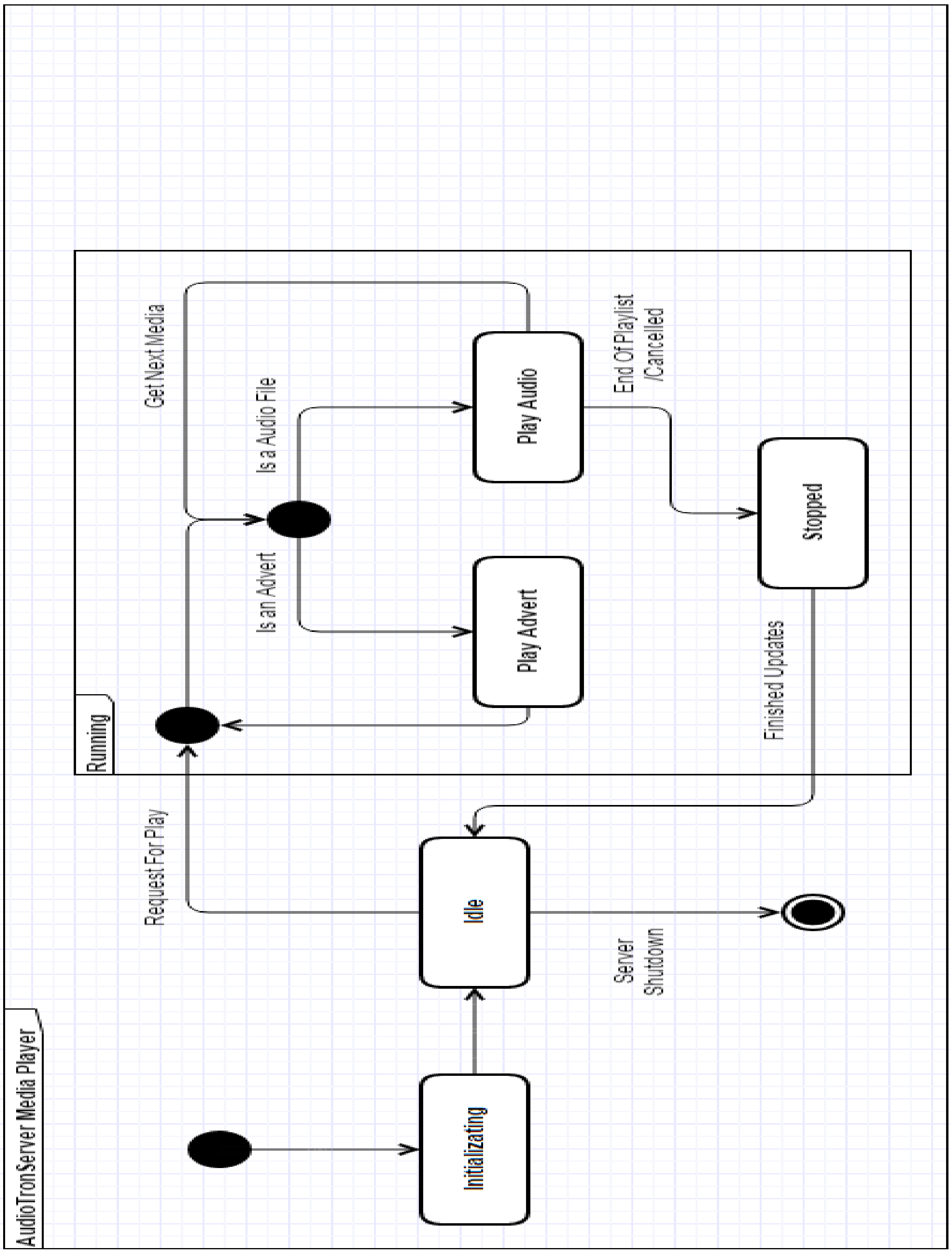
3) **Composite pattern**: The Audiotron system must provide means for users to play albums, but also allow them to open and browse albums as media folders. By applying the composite design pattern both albums and audio inherit from mediaFile. The transparent route ensures a client can act on sub classes of mediaFIle homogeneously. The album file can be used both as a media file and an object collection.

4) **Singleton pattern**: The group agreed that only one instance of the MediaPlayer and InformationManager classes need to be instantiated at run time (primarily to avoid instantiation overhead). This goal has been achieved by applying the singleton design pattern to such classes.

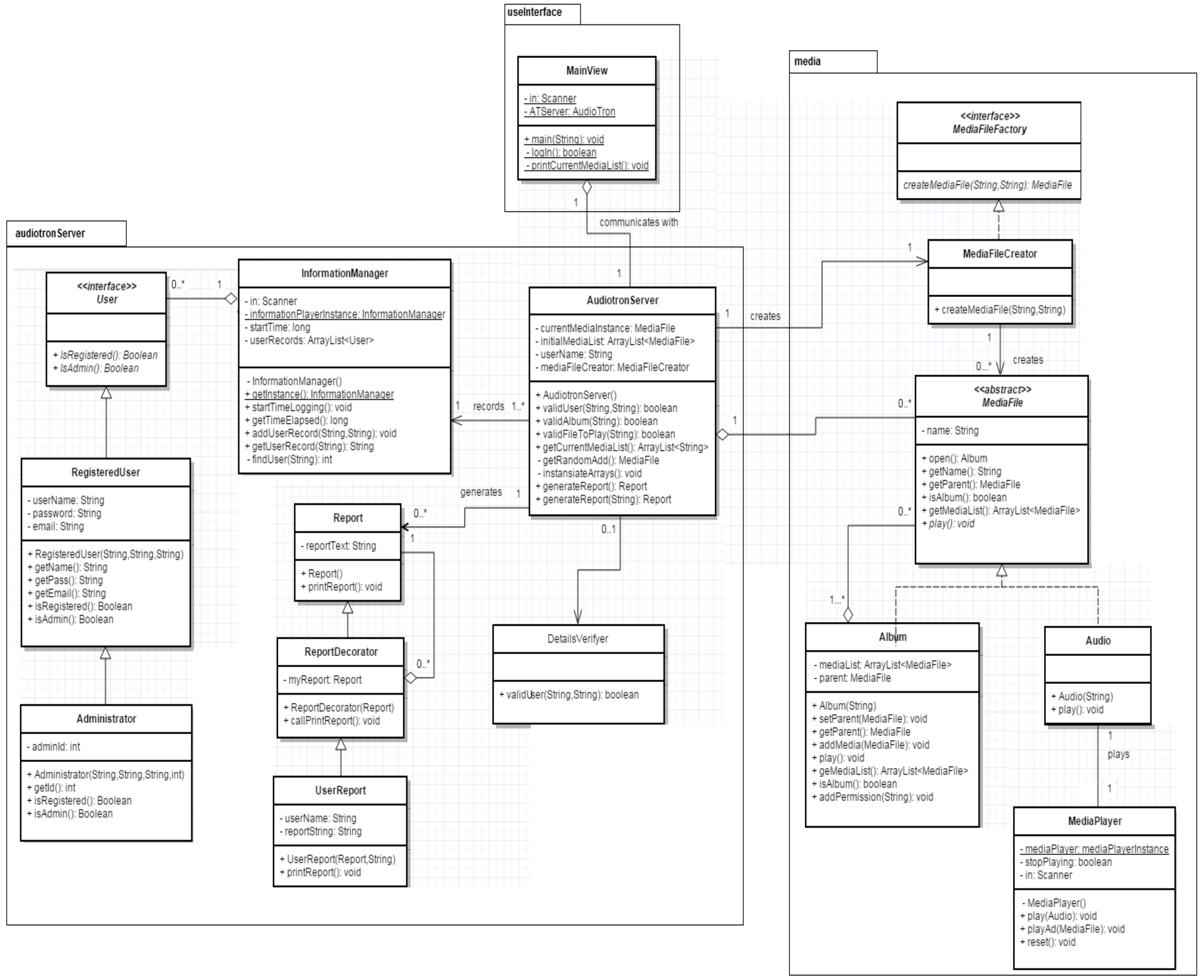
Sequence Chart: Play Media File



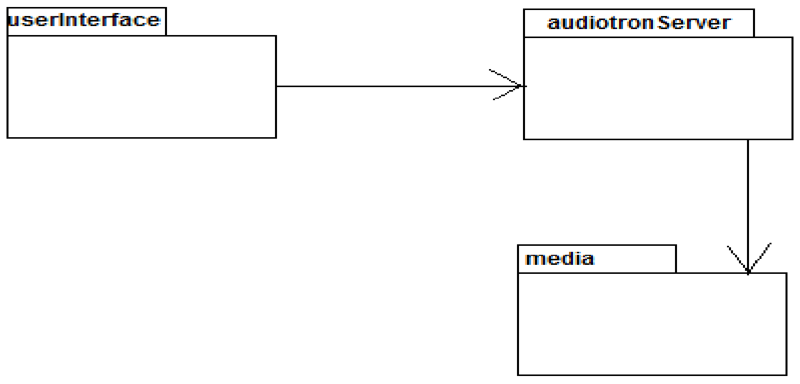
State Diagram: MediaPlayer



Class Diagram


useInterface
MainView
- in: Scanner
- ATServer: AudioTron
+ main(String): void
- logIn(): boolean
- printCurrentMediaList(): void
communicates with
audiotronServer
<<interface>>
User
+ isRegistered(): Boolean
+ isAdmin(): Boolean
InformationManager
- in: Scanner
- informationPlayerInstance: InformationManager
- startTime: long
- userRecords: ArrayList<User>
- InformationManager()
+ getInstance(): InformationManager
+ startTimeLogging(): void
+ getTimeElapsed(): long
+ addUserRecord(String,String): void
+ getUserRecord(String): String
- findUser(String): int
AudiotronServer
- currentMediaInstance: MediaFile
- initialMediaList: ArrayList<MediaFile>
- userName: String
- mediaFileCreator: MediaFileCreator
+ AudiotronServer()
+ validUser(String,String): boolean
+ validAlbum(String): boolean
+ validFileToPlay(String): boolean
+ getCurrentMediaList(): ArrayList<String>
- getRandomAdd(): MediaFile
- instansiateArrays(): void
+ generateReport(): Report
+ generateReport(String): Report
records
generates
creates
RegisteredUser
- userName: String
- password: String
- email: String
+ RegisteredUser(String,String,String)
+ getName(): String
+ getPass(): String
+ getEmail(): String
+ isRegistered(): Boolean
+ isAdmin(): Boolean
Administrator
- adminId: int
+ Administrator(String,String,String,int)
+ getId(): int
+ isRegistered(): Boolean
+ isAdmin(): Boolean
Report
- reportText: String
+ Report()
+ printReport(): void
ReportDecorator
- myReport: Report
+ ReportDecorator(Report)
+ callPrintReport(): void
UserReport
- userName: String
- reportString: String
+ UserReport(Report,String)
+ printReport(): void
DetailsVerifyer
+ validUser(String,String): boolean
media
<<interface>>
MediaFileFactory
createMediaFile(String,String): MediaFile
MediaFileCreator
+ createMediaFile(String,String)
creates
<<abstract>>
MediaFile
- name: String
+ open(): Album
+ getName(): String
+ getParent(): MediaFile
+ isAlbum(): boolean
+ getMediaList(): ArrayList<MediaFile>
+ play(): void
Album
- mediaList: ArrayList<MediaFile>
- parent: MediaFile
+ Album(String)
+ setParent(MediaFile): void
+ getParent(): MediaFile
+ addMedia(MediaFile): void
+ play(): void
+ geMediaList(): ArrayList<MediaFile>
+ isAlbum(): boolean
+ addPermission(String): void
Audio
+ Audio(String)
+ play(): void
plays
MediaPlayer
- mediaPlayer: mediaPlayerInstance
- stopPlaying: boolean
- in: Scanner
- MediaPlayer()
+ play(Audio): void
+ playAd(MediaFile): void
+ reset(): void
1
0..*
1..*
0..1
1...*
0...*

Package Diagram



Concurrency support

The Audiotron system can support concurrency through the use of the broker architecture design. As mentioned multiple times, Common Object Request Broker Architecture (CORBA) is built around this design. CORBA is often implemented in distributed system as ours would likely become (many clients to many servers), examples include the Banking and finance, Healthcare and telecommunications industry.

This architecture utilizes broker to make the server and client think they are making local calls. An advantage to this is that the broker can schedule calls from both sides at it sees fit. This scheduling would introduce the support for concurrency our system would eventually require.

At a local level concurrency can be introduced in the client and server. For instance the clients mediaplayer can queue mediaFile to play whilst the user performs other actions. Similarly generating reports may require a significant portion of time, it may be possible to thread report generation. Lastly there's clear support for task parallelism on the server side, during start up when user and media objects are instantiated both the respective factory methods handling their creation could be multi-threaded.

One must also consider the new features provided by the new Java 8 Concurrency API such as the ExecutorService. Winterberg (2015) states: “Executors are capable of running asynchronous tasks and typically manage a pool of threads, so one does not have to create new threads manually. All threads of the internal pool will be reused under the hood for revenant tasks, so many concurrent tasks can be executed throughout the life-cycle of our application with a single executor service”.

Critiques

- The original implementation consisted of a system focused on a wider spread of user functionalities. Such implementation was focused on realizing most of the use cases. On later iteration however, more focus was applied in incorporating design patterns thus reducing class coupling and allowing the selected features for implementation to run more efficiently.
- Originally many classes were implemented, each with its own function. Such implementation did not favour extensibility or reusability as a programmer who did not work on the original system would have to spend time understanding the various function calls used in multiple classes.
- During the first iteration, Advertisement was that an ambiguous entity (it was not clear which type of medium it would be i.e. audio, visual, ect.). Eventually it was decided that making advertisements a type of audio message was the most appropriate course of action (since music was the purpose of the site). Because of the similarity between advertisements and the audio classes a decision was made to make advertisements become a part of the audio class.
- Originally the system was created with a single identifiable pattern, the Factory method to create user specific reports. While this was a good implementation to achieve a flexible system for generating unique reports, the second iteration focused on implementing the decorator design pattern for reports generation. Using the decorators to model the information for reports results in a more informative and adaptable system.

- Due to time constraints the group decided to focus on functionality of the server rather than security. The skeleton classes intended for users and administrators are provided. Later iterations will be focused on implementing such classes so that priviliges and permissions can be applied where appropriate.
- The actual implementation uses text files as information storage. Future iterations will focus also on implementing database as a mechanism for the secure storage of data. The implementation of a database would lead to a more scalable and reliable system.
- The name chosen for the Album class is slightly ambiguous. The Album class is one of the components in the composite design pattern we included. The name does not indicate clearly that it can be played as well as storing other media file objects. The name playlist was considered as well, but we concluded that it did not allow for expansion of media types or indicate a hierarchical structure as an album provides.
- In the first iteration of code, in the essence of demonstrating how the system would function, the server was instantiated in the client. The current server class' implementation has a low cohesion. Albeit a number of design patterns used to manage dependencies (see implementation of factory class for generating reports) and increase cohesion. Ideally the server class should adhere to its responsibility of establishing connections and delegating work.
- As mentioned above the current iteration of code demonstrates how the system would run. It currently does not have any threaded behaviour, despite this there are multiple areas where multi-threading can be beneficial or necessary. The server would need multithreading to allow multiple user to connect at once. This could however be solved by utilizing existing application server technology such as Glassfish which has extended support for multithreading connections, as well as load balancing and clustering. Another candidate for threading would be the media player. It could operate a message queue and play concurrently whilst the user performs other options.
- The user classes aren't used. Their eventual use would be to provide a quick manner to poll what permissions a user has, what information they must provide to log in (an admin for instance must provide an admin ID), what preferences they have selected

and perhaps statistics on their use of the service. Elements like these could be easily retrieved when generating a report.

- The singleton classes MediaPlayer and InformationManager are not thread safe. As the current implementation does not have any threading the group did not focus on providing a thread safe environment. However such classes eventually would have to be made thread safe as they are likely candidates to provide trouble on this front. In particular the Information Manager would have to get and set variables at frequent intervals.
- The current implementation does not deal with some form of object passing between server and client. In a future iteration it would be necessary to invest in a technology such as CORBA or SOAP to deal with transferring media files from server to client. Lacking this feature had the negative effect on making the code more likely to change in the future. Less code refactoring may have to take place had we chosen on what technology to use in advance.

References

- International Federation of the Phonographic Industry (2015) *Digital Music in figure* [online], available: http://www.ifpi.org/facts-and-stats.php [accessed 19 November 2015].
- US Department of Veteran Affairs (2015) *System Design Document Template* [online], available: www.va.gov/osdbu/docs/news20140218attachment2.docx [accessed 15 October 2015].
- Winterberg, B. (2015) *Java 8 concurrency tutorial: Threads and Executors* [online] available: http://winterbe.com/posts/2015/04/07/java8-concurrency-tutorial-thread-executor-examples/ [accessed 27 November 2015].

List of Illustrations

- Illustration 1: Flock of Birds (2015) *Agile Development Diagram* [image online], available: http://www.flockofbirds.nl/summer-training/ [accessed 19 November 2015].
- Illustrations 2 – 3: Jamendo Music (2015) *login form* [screenshot online], available: https://www.jamendo.com/start [accessed 19 November 2015].
- Illustration 4 (adapted from): Visual Paradigm (2015) [image online], available: http://www.visual-paradigm.com/tutorials/seqanimacian.jsp [accessed 15 November 2015].
- Illustration 5 (adapted from): IVOA (2015) [image online], available: http://wiki.ivoa.net/bin/view/IVOA/RWP04SequenceDiagrams1 [accessed 15 November 2015].

List of MediaFile

- The incredible Machine 3 (1994)
  - 60's rock
  - 1959 prom
  - Bongo Bango
  - Detective theme
  - Hay seed
  - Hip Hop
  - Huey dewey
  - Keep tryin
  - New Age
  - Pictures
  - Progressive

    - Rag Time
    - Sierra intro
    - Techno Rave
    - TIM
    - Tuna Loaf
    - Unplugged

- VGMusic.com
    - Actmix
    - BBTechno
    - Bloody tears
    - Celes
    - Cr-Itoi
    - Drugstore
    - Kasandora
    - Kiwi-g
    - Punch out

- Castlevania 2 (1987)
    - Silence of Daylight