



SYSC 3010

Fall 2014

[Team Nile – Term Project Final Report]

Members: Noah Kipin, Brandon To, Nhat Ho, Itaf Joudeh, Haifa Aljasser

1. Introduction

In the interest of the Discovery Centre in the MacOdrum Library at Carleton University, Dr. Alan Steele has approached the class with a request for project proposals. He wanted a group to take on a project for a projector control system for a room in the library. This room has three projectors mounted to the ceiling pointing in the directions of three white walls and has speakers that support 7.1 digital surround sounds (see Figure 1 below). The idea is to use this system to play synchronized environmental videos and sounds to meet the needs of the students who want to study there. The design and implementation is left to the engineering team, and the type of video and audio is left to the Philosophy students. We, Team Nile, accepted this project with some modifications as our term project for SYSC 3010. The system also includes an infrared (IR) remote control device to control the video and audio playback as the “hardware” requirement of the term project. The project meets the networked requirement, with three Raspberry Pi’s streaming video and audio from a centralized server. The GUI requirement was accomplished by creating a web interface for the users to control the system on their tablet.

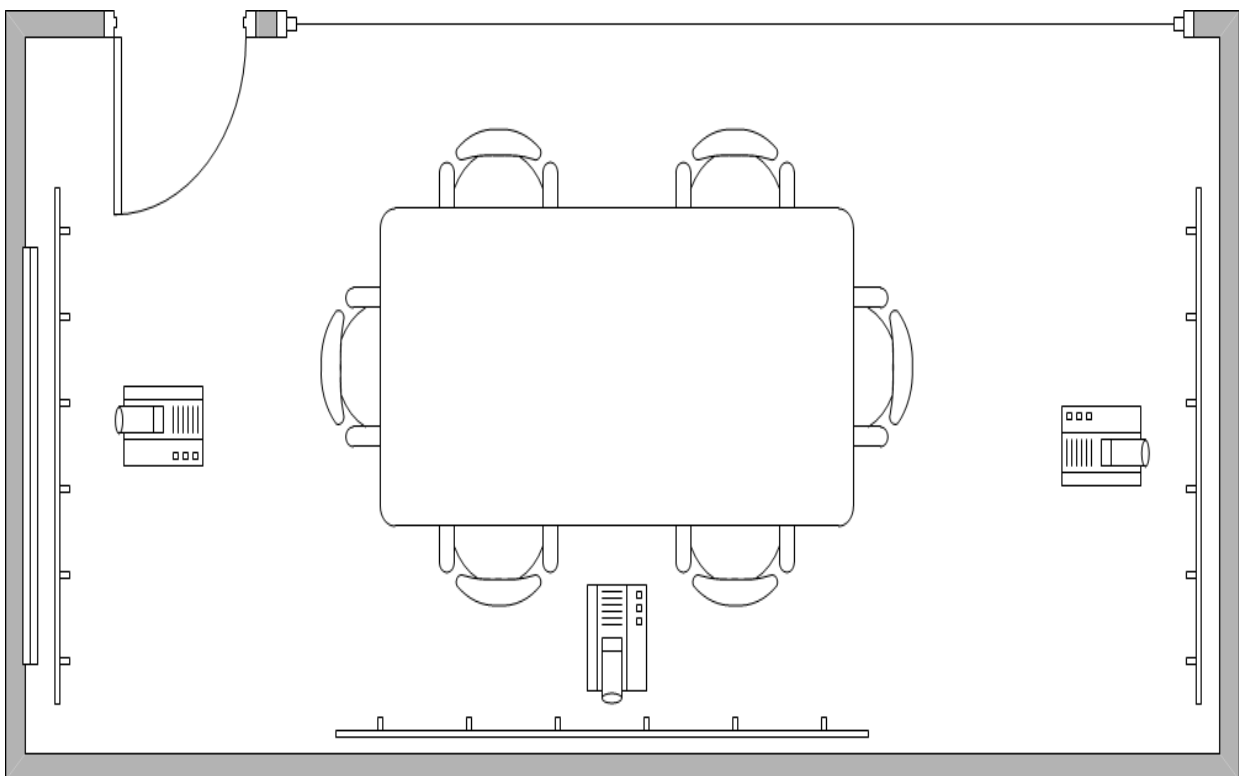


Figure 1: Representation of the Audio-Video Study Room

2. Engineering Design

All aspects of engineering design are going to be outlined in this section. This section consists of six other subsections: Problem Identification, Proposed Solution, Requirements, Design, Implementation, and Testing.

2.1 Problem Identification

The main purpose of this project is to provide an engineered solution to give users an easy to use system for controlling projectors and speakers. The system consists of three major subsystems: the Scene Manager, the IR Control, and the Content Distribution and Playback Unit.

The Scene Manager subsystem should aim to provide users with the ability to control the system through a graphical user web interface (GUWI). This subsystem should let the student users, play/pause/stop/skip, and adjust the volume of the scenes. On the other hand, it should let the admin users, view/upload/edit/delete scenes, do anything the students could do, and configure system settings. The web interface should be usable on mobile platforms (i.e. iOS, Android, etc.). A visual representation of the Scene Management subsystem is shown in Figure 2.1 below.

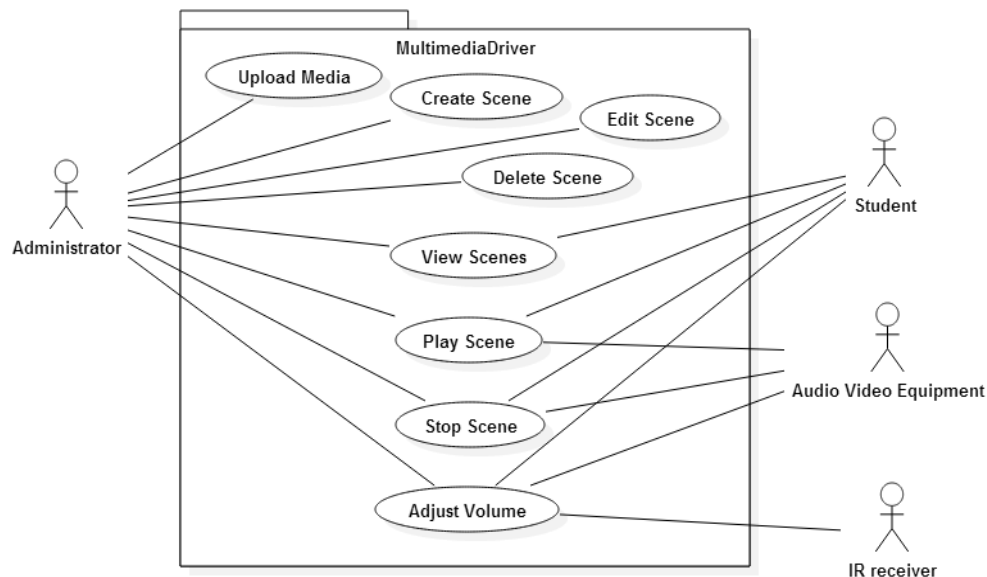


Figure 2.1: UML Use Case Diagram

The IR Control subsystem should aim to provide users with the ability to control the scenes through a physical infrared remote control. This subsystem should accept remote control input which maps to a

subset of the features that the Scene Manager Web Interface provides. These include allowing users to play/pause/stop/skip, and adjusting the volume of the scenes.

The Content Distribution and Playback subsystem should aim to provide the system with the ability to play video on the projectors and audio on the speakers. Synchronization of the video and audio is handled by this subsystem. The Content Distribution system should control three Raspberry Pi's (one Raspberry Pi per projector) providing the video and audio content to the projectors. The centralized server should house all of the uploaded content. The Raspberry Pi's should act as the clients that stream video and audio from the centralized server.

2.2 Proposed Solution

This section includes detailed explanations of our proposed solution. Figure 2.2 shows a UML Class Diagram of the three subsystems and the different proposed classes for each of them.

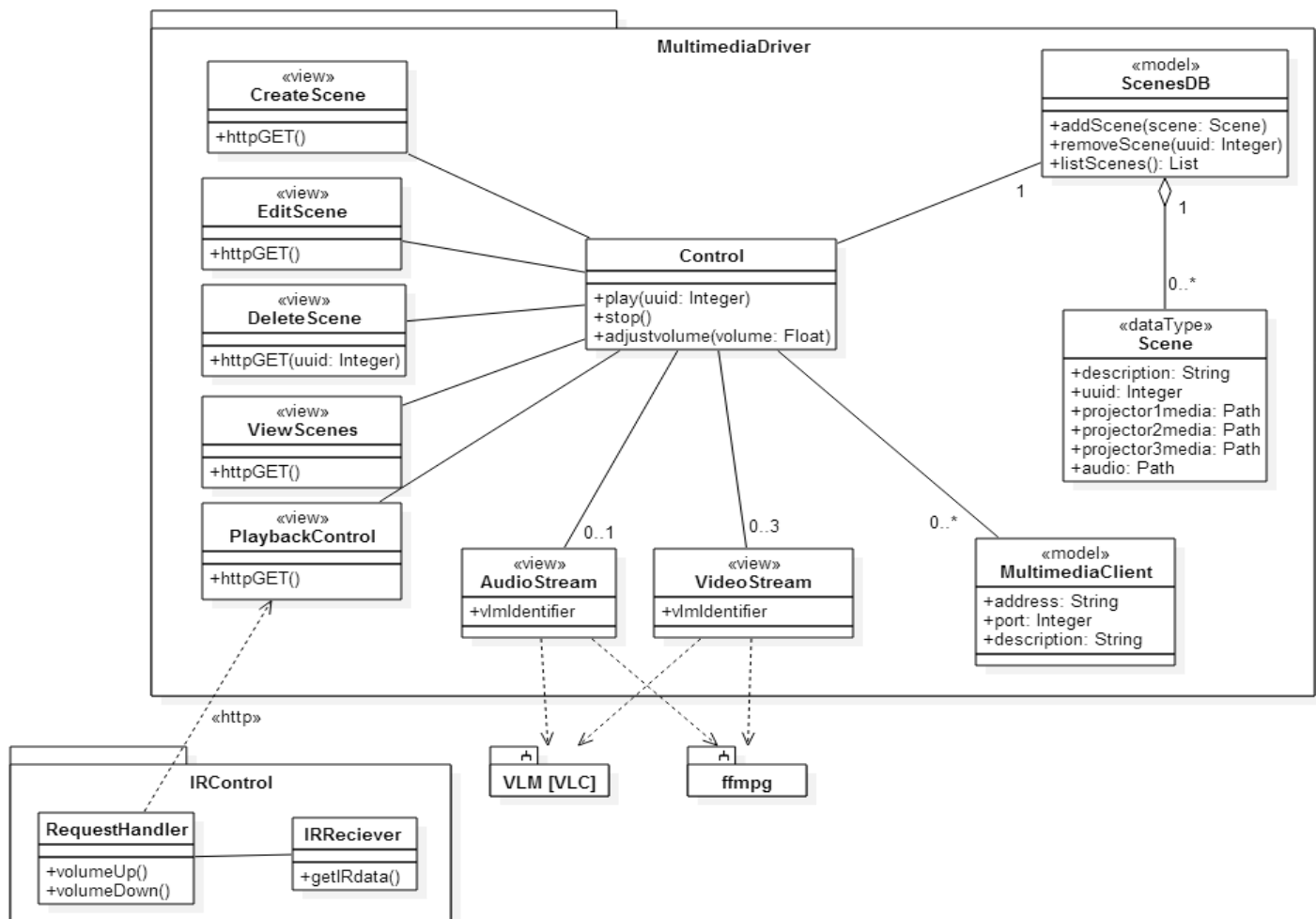


Figure 2.2: UML Class Diagram

2.2.1 Scenes Management

A web service which provides webpages to create, manage, and control playback of the scenes was created to accomplish the scene manager subsystem. A scene is a selection of video and audio files which should be played concurrently. Each scene will designate a piece of media to be played through each client of the Content Distribution system (a piece of media will need to be selected for each of the three video clients and the audio client). Once scenes are created they can be selected for playback through the scene management system. When a scene is selected for playback the scene management system will communicate with the content distribution system, providing the distribution system with the scene information (location of each media file to be played and the places they should be played). The scene manager will also be able to be used for playback control. All the functions of the scene manager will be able to be accessed through modern browsers (including iOS or Android based mobile devices).

Asynchronous JavaScript and XML (AJAX) is a method in building a web interface where the client's requests are handled immediately. This method will be used in combination of several programming tools and languages such as: JavaScript, dynamic HTML (DHTML), Extensible Markup Language (XML), cascading style sheets (CSS), the Document Object Model (DOM), Microsoft Object, and XML HTTP Request. AJAX allows the immediate response from the server on the client's actions; this enables the graphical user web interface to be interactive and real-time. This method will be used in creating the webpages; it allows the requests from the users to be processed on the server without delay.

Database MySQL is a relational database management system (RDBMS); this system is based on Structured Query Language (SQL). SQL is a language which is used to communicate with a database. MySQL is quite flexible in usability; however, it is mostly used in web-based applications and online publishing. The reason is that MySQL provides multiple web-optimized features such as HTML data types: FLOAT, VARCHAR, BINARY, BLOB, DATETIME, TIMESTAMP, etc.

MySQL will be used in the system as the database which will store the audio, video files and the scenes.

2.2.2 IR Control System

An Infrared (IR) Sensor is an electronic instrument that is used to sense certain characteristics of its surroundings by either emitting and/or detecting IR radiation. In the IR Control system, an IR receiver will be connected to a Raspberry Pi through a PiFace or directly. The receiver will detect the coded signals sent by an IR controller. After that, the decoded signal will be sent to the scene management system to perform accordingly.

In other words, the IR Control system will be created by having a monitoring daemon on a Raspberry Pi. The monitoring daemon will read an IR receiver connected to the Raspberry Pi. When the monitor daemon senses IR Control signals (i.e. from a universal remote control) the daemon will communicate through HTTP with the scene management system to play playback, stop playback, increase volume, or decrease volume.

2.2.3 Content Distribution

The content distribution system will control audio video clients (Raspberry Pi's) and assign them to stream an audio or video file. The content distribution system will also synchronize the playback so that each media streaming client starts playback at the same moment which will provide a seamless experience. The content distribution will be facilitated by communicating with the client player's HTTP interface. The media to be played will be queued on each device, and then a signal will be broadcast to begin playback. When the clients receive the signal to begin playback they will start streaming the media designated for them by the content distribution system from the locations outlined in the scene.

Real-Time Transport Protocol (RTP) is an Internet protocol standard that specifies a way for programs to manage the real-time transmission of multimedia data over unicast or multicast network services. This protocol will be used in the content distribution system. It ensures that the system can

assign tasks to multimedia clients (Raspberry Pi's) and synchronously start performing the assigned tasks in each client.

VLC Media Player is a portable, free and open-source, cross-platform media player and streaming media server. VLC includes multiple coding and decoding libraries. Most importantly, VLC provides an HTTP Lua module which enables controlling VLC through a browser interface. VLC will be used in the system to playback the audio and video files.

2.3 Requirements

2.3.1 Scenes Management

To establish the Scenes Management subsystem, a graphical user web interface was created. The website requirements are as follows:

- A tablet will be used to access a site where students can select a view to be played.
- An administrator site which only allows librarians and professors to manipulate the scenes.
- Admins must be able to view the scenes that exist in the database, as well as adding, editing and deleting scenes conveniently.
- Security layers between the two sites so that students cannot access the administrator site.
- Make sure users must login before accessing the webpage, either to the student site or the admin site.

2.3.2 IR Control System

The requirements for the IR system are as follows:

- The system must be able to communicate to the web server using the hypertext transfer protocol (HTTP).
- The system must be able to detect key presses on an infrared remote control.
- The system must be able to send the key presses over a network with minimal delay.

- The system must be capable of stopping, starting, playing, pausing, back/forward skipping, controlling the volume, and powering off the projector system.
- The system must be able to run efficiently for extended periods of time.

2.3.3 Content Distribution

To establish the Content Distribution subsystem, a playback streaming control software was created. The requirements for the streaming control system are to:

- Control the streaming software to stream scenes given by the scene management component.
- Communicate with the external Raspberry Pi's to start, stop, and pause playback as well as adjusting the volume.

2.4 Design

2.4.1 Web Interface

In order to accomplish all of the above goals, the following design techniques were selected:

- The websites for both students and admins will be implemented on HTML5, and CSS3 for maximum compatibility, and scalable according to screen size.
- For smooth manipulation on the website, JQuery will be used alongside with JavaScript to reduce the complexity of codes, and to provide high performance.
- The scenes information will be stored in the phpMyAdmin tool, which utilizes MySQL languages. MySQL works quite well with PHP, which was used to implement the server of the system.
- Plupload API will be used to allow admins to upload images, videos and audios from their computers. This technique eliminates manual tasks of transferring files into the correct locations, and configuring files according to system's requirements.
- 'Hash 512' will be used to encrypt and decrypt passwords which are entered by users. This hashing technique provides an average security level in the Local Area Network (not so much

in the Internet). However, since this system is used mainly at school and through a local server, 'Hash 512' should serve well for this purpose.

- Before the website is loaded, the server needs to check if users have logged in or not. If users have not logged in, the system will redirect the users to the login page and prevent trespassing to any other pages in the system.

2.4.2 IR Control System

To fulfill the requirements listed above, the following design choices were made:

- The program will be implemented in the C programming language for execution speed.
- The program will make use of Berkeley sockets for networked communication.
- The program will implement version 1.1 of the HTTP.

The program will make use of the LIRC library (<http://www.lirc.org>) for implementation of the infrared capabilities.

2.4.3 Streaming Control

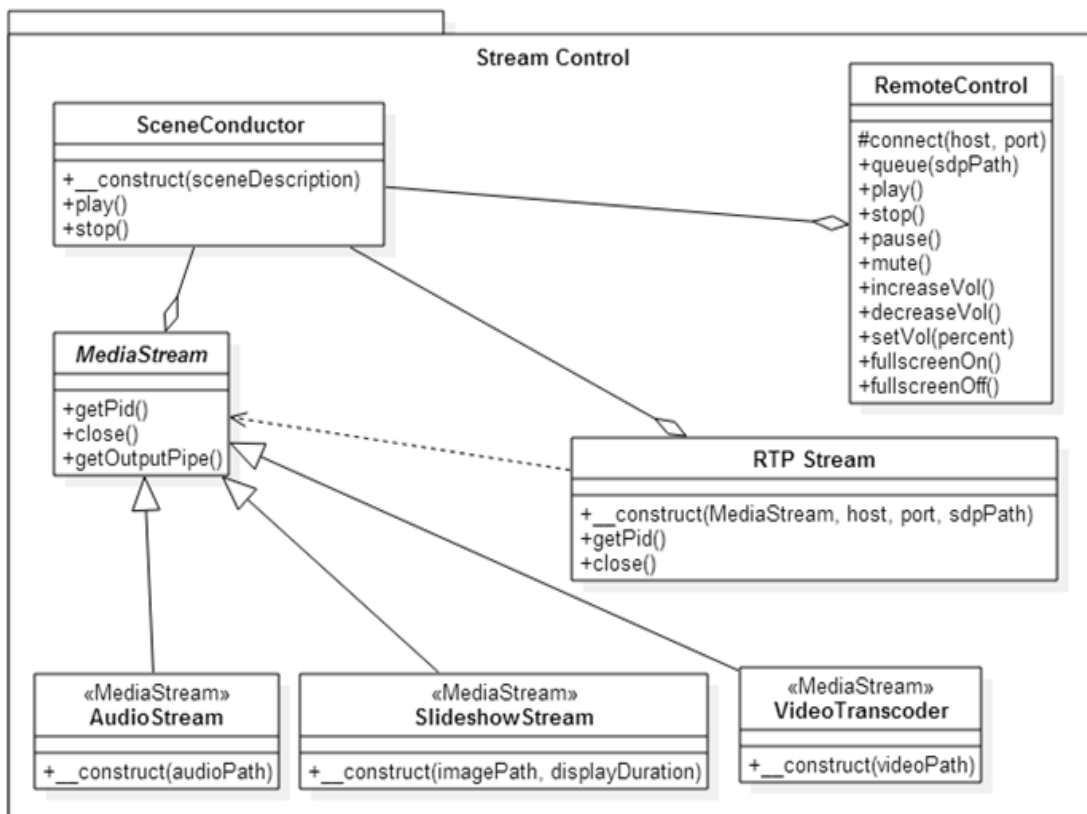


Figure 2.3: Streaming Control UML Class Diagram

The streaming control component needed to coordinate many different facets of the project, and so it needed to take a very modular approach. The Scene Conductor will be the main coordinator for all of the media playback and streaming from the server, and the RemoteControl class will handle instructing the remote Raspberry Pi client software.

SceneConductors get constructed with a description of the scene to be played. The Conductor then opens the media to be played through the appropriate MediaStream implementation: AudioStream for streaming audio tracks, SlideshowStream for constructing a video slideshow from pictures, or VideoTranscoder for video files which are not already in the H264 format. Each MediaStream's output is then passed to an RTPStream instance which uses the raw audio or video data stream and creates an RTP stream that can be sent over the local network to a Raspberry Pi running the client software.

The Remote control class is constructed for each client Raspberry Pi. The conductor is able to communicate through the RemoteControl class to instruct each Raspberry Pi to stream the correct RTP stream. This RemoteControl class is also used by other portions of the project such as the IR control component.

2.5 Implementation

2.5.1 Web Interface

The website was implemented using combination of HTML5, JavaScript, CSS, PHP, MySQL as main languages. In addition, some extra libraries and tools such as: JQuery, Plupload API, Bootstrap, Bootbox and XAMPP were used to setup the environment for testing and developing as well as extra features in the final product.

The General User Interface consists of five components: the student view, the admin view, the server, the file system, and the database. The server will be explained alongside with the student and admin views.

Login View:

In order to access the Student view or the Admin view, users would need to log in as a 'student' or as an 'admin', respectively. The login page is shown below:



The image shows a login form with a light blue background. At the top center, the text "Sign in" is displayed. Below this, there are two white input fields with blue borders. The first field is labeled "Username" and the second is labeled "Password". Below the password field, there is a checkbox with the label "Remember Me". At the bottom of the form, there is a blue button with the text "Sign in" in white.

Figure 2.4: Login View

If the users login as a 'student' they can only access the student view. On the other hand, if they login as an 'admin' they can access both the student and admin views.

1. When users input a username, and a password then click 'Sign in', the client side would retrieve the inputs, hash it using 'hash 512', and make an AJAX call to the server side.
2. The server side receives parameters from the client side including the 'username' and the hashed 'password'.
3. The Server makes a call to the database to retrieve the 'password' and the 'salt' associated with the 'username' (refer to the Database section).
4. The password which was hashed in the client side would be hashed again with 'salt'. After the second hash, the password would be compared with the 'password' from the database.
5. If the 'username' and the 'password' are correct, users will be redirected to the student view if logged in as a 'student', or the admin view if logged in as an 'admin'.
6. On the other hand, if the 'username' and the 'password' are not matched, users will be asked to try again.

Student View:

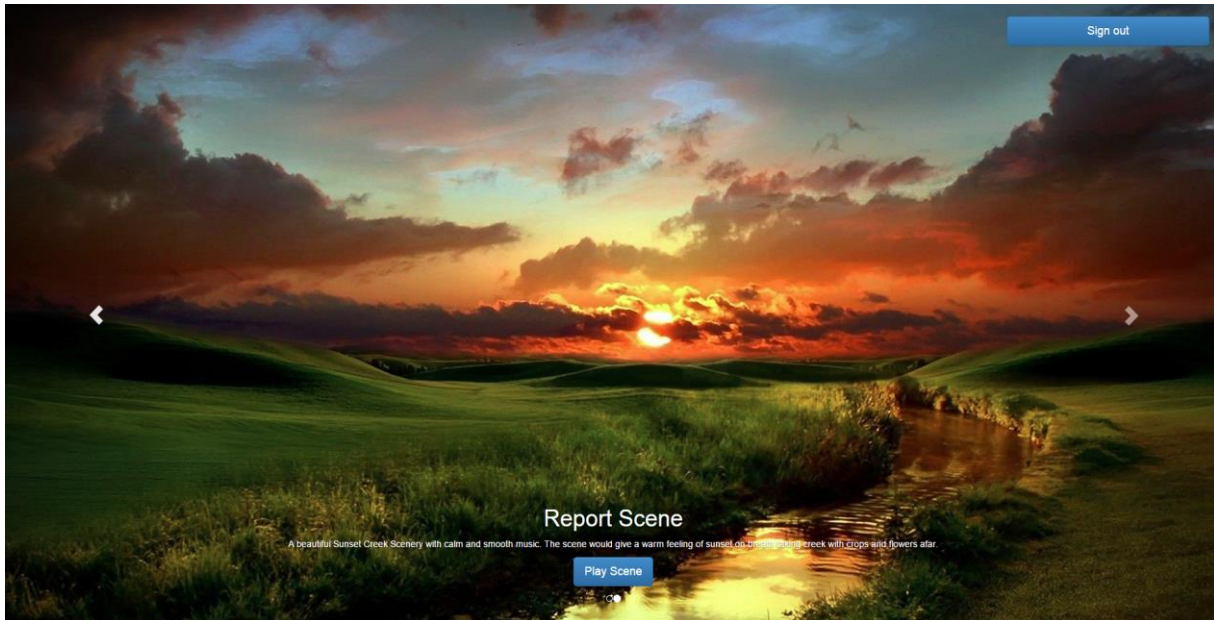


Figure 2.5: Student View

The student view is a carousel view which allows users to swipe left, and right on their tablets or phones and select a scene to be played on the projector(s). The cover page which was uploaded when creating a scene is displayed in this view. In addition, the title, a brief description and a 'Play Scene' button corresponding to this scene are also displayed.

1. The easy looking layout of this view was based on Bootstrap and JQuery.
2. Upon loading the page, an AJAX call is made to the server which requests a title, a description, and a path to the cover page of all available scenes in the database.
3. The returned response is in JSON format. The client extracts the responses and dynamically adds all of the scenes to the carousel view using JQuery. Each scene has a 'Play Scene' button corresponding to it, which contains the title of the scene as its value.
4. When users click on the 'Play Scene' button, the client would extract the value of the button which is the title of the scene. The client makes an AJAX call to the server side with one parameter containing the title of the selected scene.
5. The server uses the title to retrieve information and starts streaming as explained below in the server part.

6. When users click on the ‘Sign out’ button, the system would remove all stored information of the login session, and redirect the users to the login page.

Admin View:

The admin view consists of four different taps. Those taps are View Scenes, Add Scenes, Edit Scenes, and Delete Scenes. Only admin users such as librarians, professors, or technicians can access those taps, while student users will always get redirected to the student view if they tried to access them.

View Scenes:

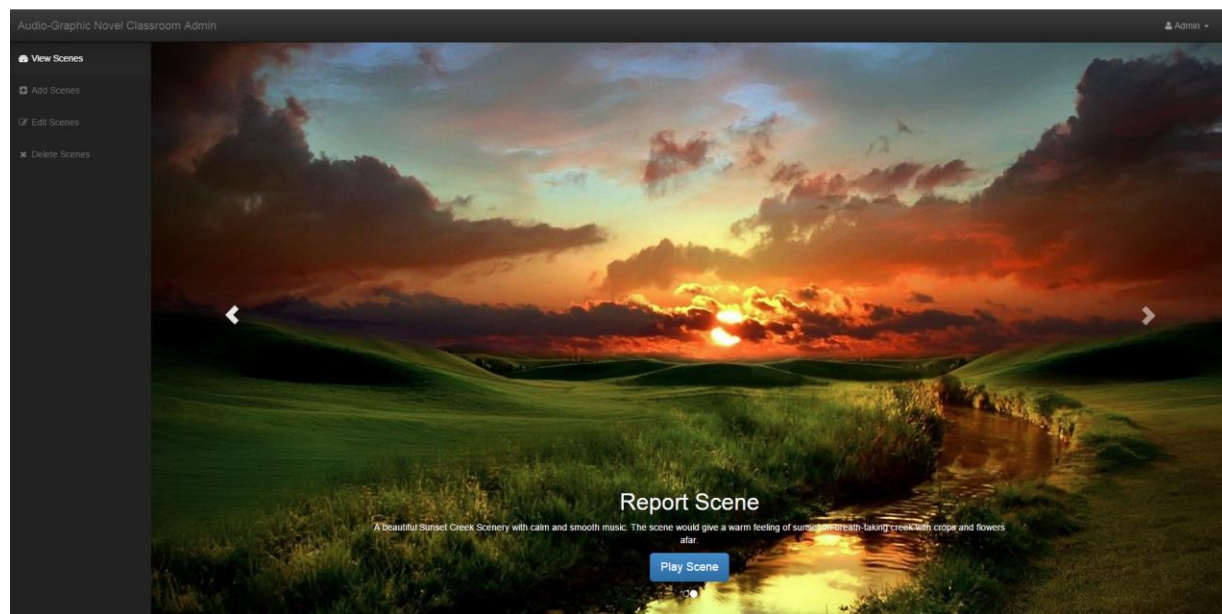


Figure 2.6: Admin View Scenes Tab

The View Scenes page in the Admin Dashboard displays the same information and provides the same functionalities as the student view. This view allows the admin to decide which scene will be played inside the study room without physically entering the room and using the tablet to change the scene. The mechanism of this tab is the same as the Student View discussed above.

Add Scenes:

The screenshot shows the 'Add Scenes' page in the 'Audio-Graphic Novel Classroom Admin' interface. The sidebar on the left contains navigation links: 'View Scenes', 'Add Scenes' (highlighted), 'Edit Scenes', and 'Delete Scenes'. The main content area is titled 'Add Scenes' and includes a sub-header 'Scene Identification'. Below this, there are two text input fields: 'Scene Title' (with an example 'Fuji Mountain') and 'Scene Description' (with an example 'which subject should this scene be suitable'). Underneath these are five green horizontal bars representing uploaders: 'Cover Image', 'Projector 1 Customization', 'Projector 2 Customization', 'Projector 3 Customization', and 'Soundtrack Selection'. At the bottom of the form are two buttons: 'Create Scene' and 'New Scene'.

Figure 2.7: Admin Add Scenes Tab

The Add Scenes page utilizes Bootstrap, Bootbox, JQuery and especially Plupload API to enable easy scene creation, and dynamic file uploading feature. Each scene must have seven components: title, description, cover image, projector 1 files, projector 2 files, projector 3 files and a soundtrack.

In order to prevent the corruption of data, some restrictions were used to prevent invalid data such as:

- The title cannot contain any special characters. The only acceptable characters are: a-z, A-Z, numbers, and spaces between words. The reason for that is that the title is used as a folder's title in file system, any special character will not be allowed in folders' titles, which will create different folder's name from the one stored in database.
- The Cover Image, Projector 1 Customization, Projector 2 Customization, Projector 3 Customization, and Soundtrack Selection are uploaders that were implemented using Plupload API.

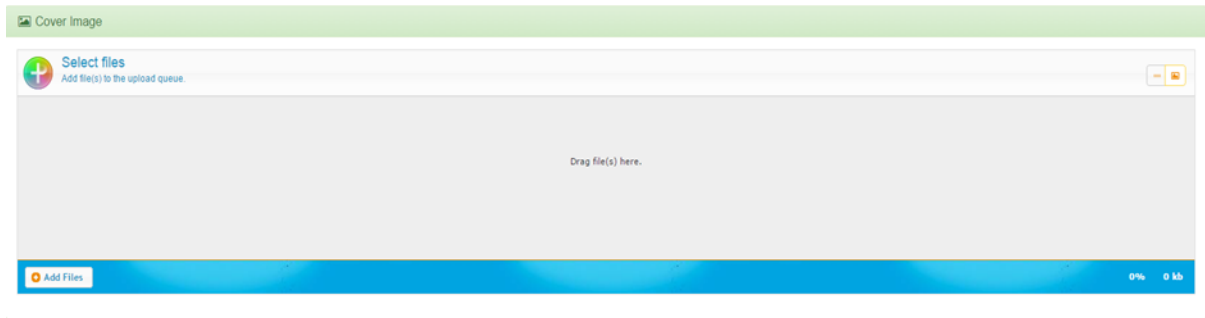



Figure 2.8: Uploader using Plupload API

- There are three types of uploaders in the system: the cover image uploader, the projector uploader, and the soundtrack uploader. Users can either click on Add Files to manually add new files or drag and drop the files.
 - The cover image uploader only allows ONE image file (jpg or png) to be uploaded. The uploader will automatically remove extra files that users attempt to upload.
 - The projector uploader is more flexible and complicated; it allows users to upload multiple image files (either jpg or png) or ONE video file. This restriction is due to the limitation on the streaming mechanism in the server. Multiple image files can be put into a slideshow which will be streamed by the server (this is not supported at the moment). The uploader will automatically remove image files which do not have the same extension of the first file in the queue (i.e. if the first file is .jpg file, the rest must be .jpg files). Which means the projector uploader prevents uploading multiple file types, and also removes extra files if the first one is a video file.
 - The uploader will also resize the images to 1920 x 1000 pixels for the best display. Because of this, it is recommended to upload images with higher resolutions than 1920 x 1000 pixels.
 - The soundtrack uploader only allows users to upload ONE mp3 file up to 3GB in size. Any extra files will be removed automatically when users attempt to upload them.
- When users click on the 'Create Scene' button:

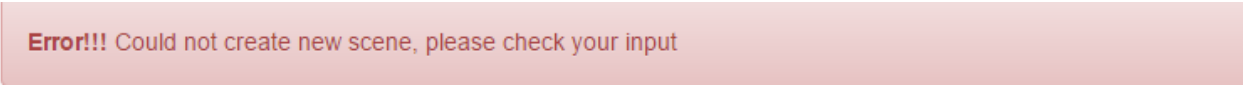
- The client side would check if the title and description fields are entered or not, and whether each uploader has at least ONE valid file in it or not.
- After that, the client will display a confirm message using Bootbox which contains the contents of the title and description fields, and the names of all files in each uploader.
- When users click “Confirm”; the title, description, number of files, and files’ names are submitted to the server to be processed and stored in the database (refer to the Database part).
- Once the submitted data are processed, the server would send back a success message to the client in JSON format.
- The client extracts the JSON message and decides if the form was successfully submitted or not. If the form was successfully created, the client will start the uploading processes of all five uploaders.
- After that, the client displays a success message to let users know that a new scene was created.



Well done! You successfully added a scene

Figure 2.9: Success Message in Creating Scenes

- If the JSON response indicates that the scene cannot be created, due to same title or invalid inputs, a failure message will be displayed.



Error!!! Could not create new scene, please check your input

Figure 2.10: Failure Message in Creating Scenes

- Because of the slideshow feature’s limitation: the system can only stream files in the folder with a common name, the images in projectors’ uploaders need to be renamed to “image1.jpg,” “image2.jpg,” etc. or “image1.png,” “image2.png,” etc. To achieve this, after all uploaders have uploaded the files, the server would perform files re-organization inside the file system every time the page is loaded. The file “indicator.txt” in the file system is used to set a flag that

indicates whether a folder needs to be re-organized or not. If the content of the file is “Edited”, the server will clean up the folder and change the content to “Organized”. Add Scenes and Edit Scenes are the two places where the “Edited” flag will be set.

- When users click on New Scene, the Add Scenes page will be reloaded and all the input fields as well as the uploaders will be emptied for new inputs.

Edit Scenes:

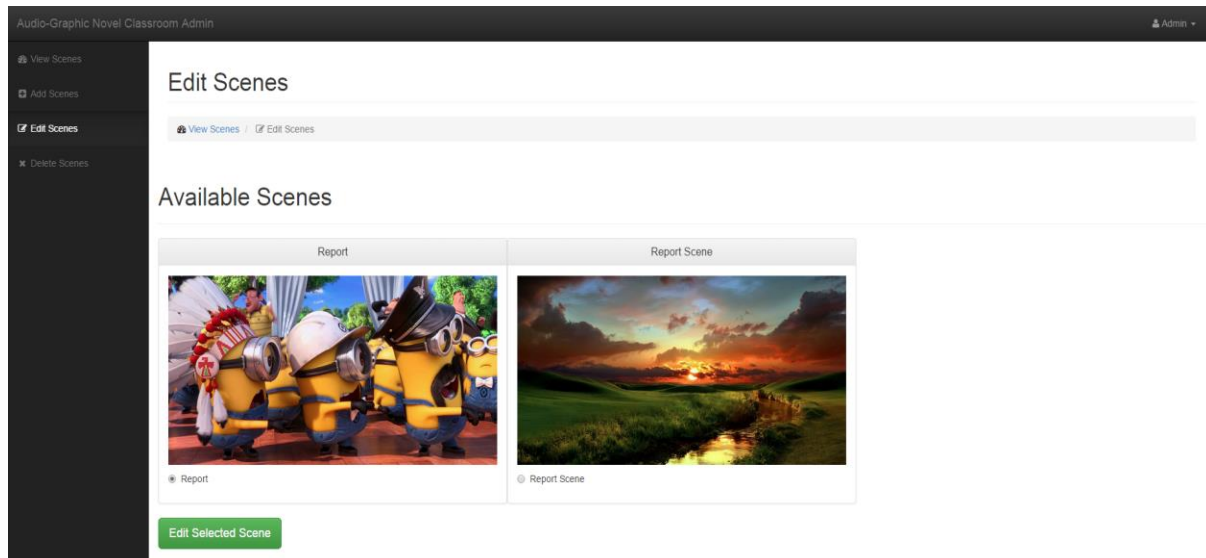


Figure 2.11: Admin Edit Scenes Tab

The Edit Scenes page displays the collection of all available scenes in the database.

- Upon loading the page, the client side sends an AJAX request to the server to retrieve information including: the cover image and the title of all scenes.
- Each scene is put into a frame as shown above.
- The admin is restricted to edit only one scene at a time. The radio button in each scene is a member of the view’s radio group, which enforces the selection of one scene.
- When the admin selects a scene by clicking on the radio button associated with that scene, and clicks on “Edit Selected Scene”, the client will retrieve the value of the radio button which is the title of the scene. The client then makes an AJAX call to the server and passes the title as a parameter.

- The server uses the scene's title to retrieve all the information including: the title, the description, the number of files, the files' names in each projector, the cover image name, and the soundtrack name.
- The server packs all of the above information into JSON format and returns to the client side.
- The client side opens a similar form to the Add Scenes page with pre-populated data of the current scene (as shown in Figure 2.12 and 2.13). The title, however, is set to read-only, this is done to prevent complication in the database and the file system.

Figure 2.12: Edit Scenes Form

Figure 2.13: Pre-populated Uploader View

- The pre-populated data in each uploader are not references to actual data in the file system. Because of security reasons, the client side should not have direct access to the file system. In order to bypass this restriction, Plupload File objects are created using files' names retrieved

from the database. These file objects can be removed from the upload queues and still enforce the uploaders' rules discussed in the Add Scenes section above.

- The admin can add more files or remove the current files in the uploader's box either using the 'Add Files' button or the Drag and Drop feature.
- When the admin clicks on Edit Scene, the client side would verify that each uploader has at least ONE file and that the description is not empty.
- After that, the client will display a confirm message using Bootbox that contains the title, the description, and the files' names in all uploaders.
- Once the admin clicks confirm, the form will be submitted. The server receives the parameters and changes the contents of the database accordingly using the "UPDATE" command in MySQL.
- The server sets the "indicator.txt" to "Edited" so it can clean up the folder later. After that, the server cleans up the old files that were removed by the admin in the file system. The server uses the files' names string in each uploader, and compares it with each file in the file system. The files in the file system that do not appear in the files' names string are then removed.
- The server returns a success JSON response to the client.
- The client receives and decodes the JSON response. After that, it initiates uploading new files in all of the uploaders. Then, it displays a success message similar to the one associated with the Add Scenes page.
- After successfully editing a scene, the Edit Scenes page is reloaded and the new data is loaded into the client view.
- If the admin clicks on Cancel Form, the form will disappear and the page is reloaded to get a fresh set of data from the server side.

Delete Scenes:

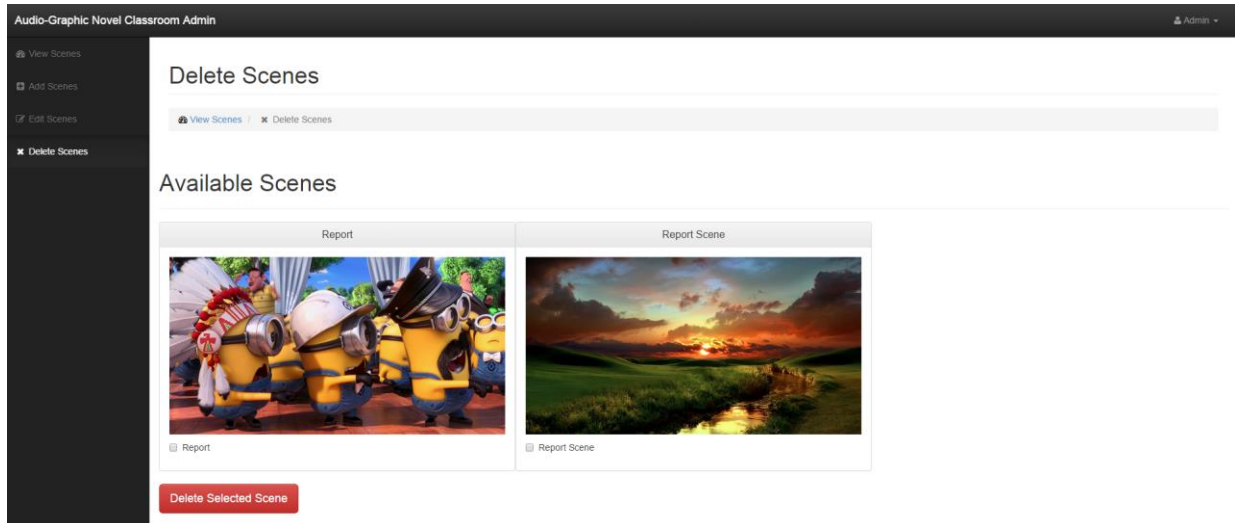


Figure 2.14: Admin Delete Scenes Tab

The layout of this page is quite similar to the Edit Scenes page. In contrast, each scene has a checkbox which is a member of the checkbox group of the page.

- The admin can select a single or multiple scenes to be deleted.
- If no scene was selected and “Delete Selected Scene” is clicked, a warning message will appear:

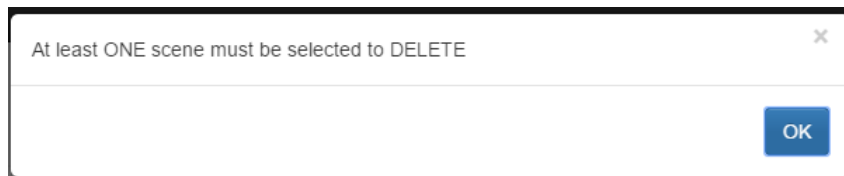


Figure 2.15: Warning in Delete Scenes Page

- When scenes are selected and “Delete Selected Scene” is clicked, a confirmation message which was implemented using Bootbox will appear containing the name of all selected scenes:

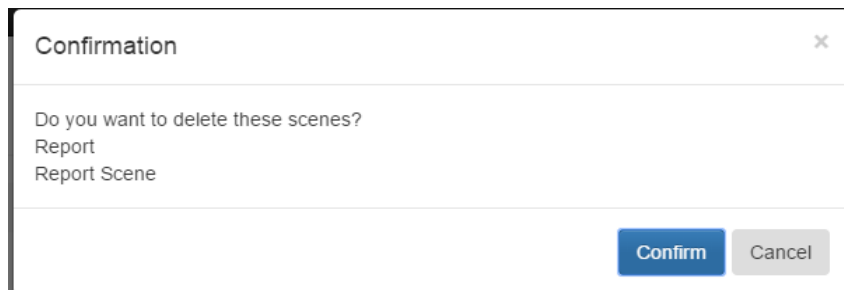


Figure 2.16: Confirmation in Delete Scenes Page

- Once the admin clicks confirm, the client puts all of the selected scenes' names into a string and makes an AJAX call to the server.

- The server retrieves the string and splits it up into an array of scenes' names.
- The server loops through the array and deletes the rows containing the scenes' titles in the database using the "DELETE" command in MySQL.
- After that, the server deletes the folder which contains the scene in the file system. Upon completion, the server would return a success JSON message to the client. The client would then reload the page to get a new set of data from the server.

No Available Scenes:

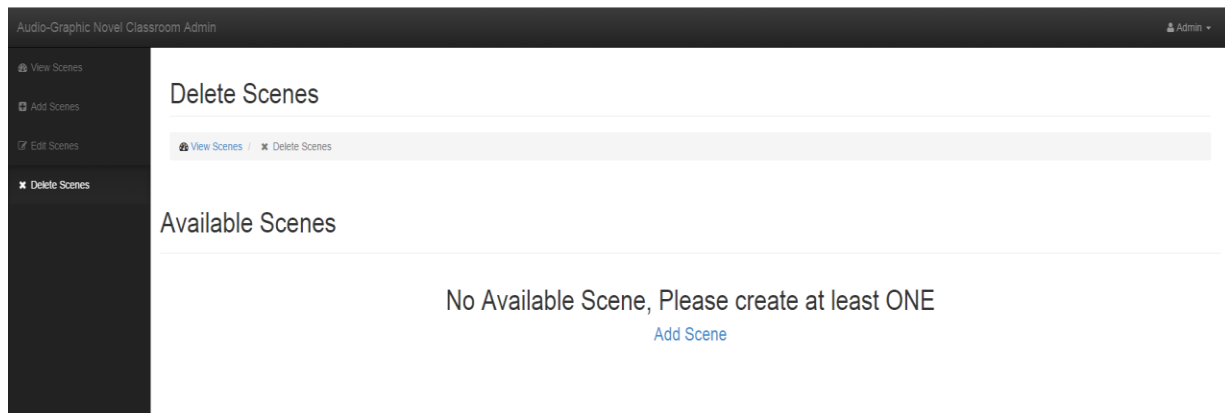


Figure 2.17: Admin No Available Scenes View

When there is no scene to be view, the client side will prompt the admin to the Add Scenes page, so new scenes can be created and used.

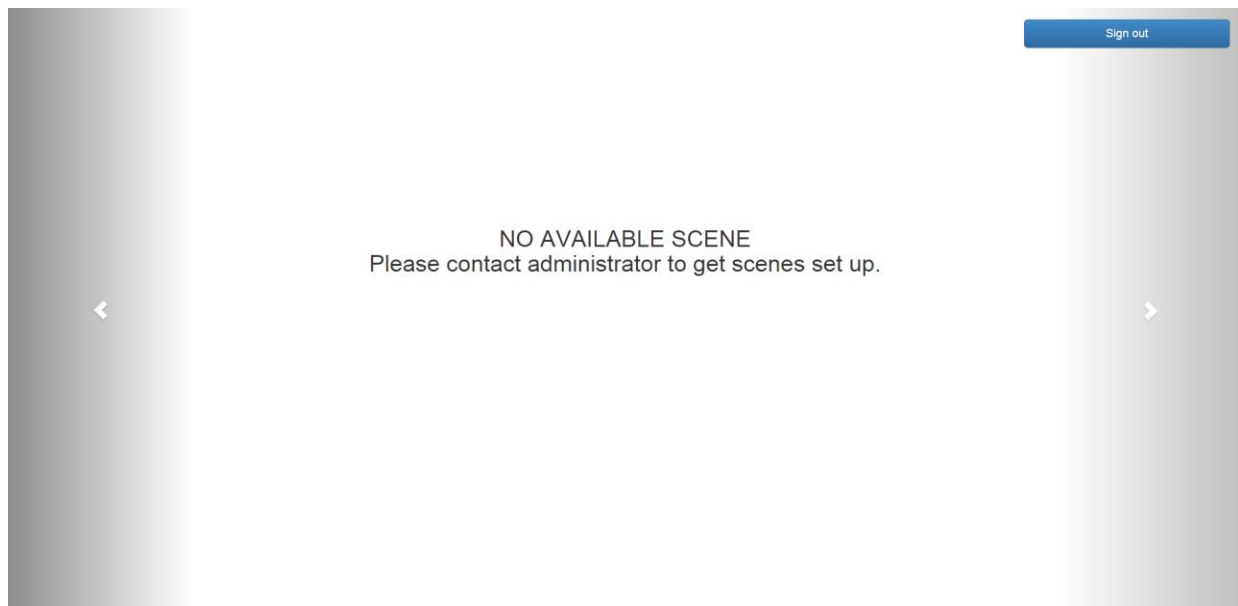


Figure 2.18: Client No Available Scenes View

Similarly, when there are no scenes available in the client side, the webpage prompts users to contact an administrator to set up scenes. Since it is prohibited for users to access the Add Scene page, this seems to be the only option the students have.

The File System:

Name	Date modified	Type	Size
1	2014-12-07 12:55 ...	File folder	
2	2014-12-07 12:55 ...	File folder	
3	2014-12-07 12:55 ...	File folder	
Hat Voi Thien Than - Chua ro.mp3	2014-12-07 12:55 ...	MP3 Format Sound	3,507 KB
indicator.txt	2014-12-07 12:55 ...	Text Document	1 KB
sunset-creek-scenery.jpg	2014-12-07 12:55 ...	JPEG image	350 KB

Figure 2.19: File System Structure for Report Scene

Each scene once created will have six elements in a folder named after the scene title in the File System. The folder contains three subfolders: 1, 2, and 3 which contain the contents to be shown by Projector 1, 2 and 3 respectively. The .mp3 file is the soundtrack and the .jpg file is the cover image. The indicator.txt file is used as a flag to let the server know whether this scene was newly created or has been recently edited.

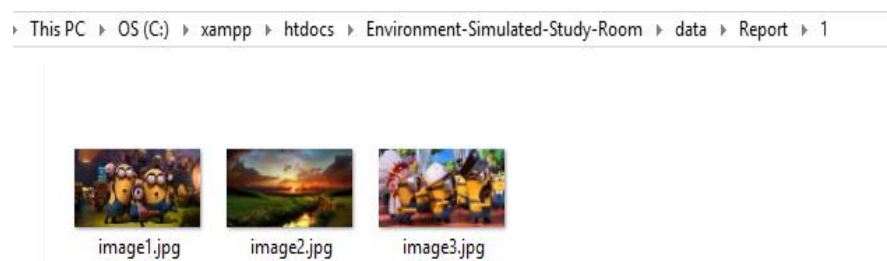


Figure 2.20: Renamed Images

As mentioned in the Add Scenes section, due to the server's limitation in the streaming slideshow, all images in folders 1, 2, 3 will be renamed when added or edited.

- When adding or editing scenes, after the form is submitted, the indicator.txt content will be set to “Edited”.
- The server will check the file of all the scenes when it is loading and re-organize the scene if “Edited” is detected.

- The server will then change the contents to “Organized” so that the next time it loads the page it will skip the organizing process and boost performance.

The Database:

Users table:

+ Options

username	password	salt
admin	7fe2058362c24e1f13b64dd2fb6206854e6a4f94b935b632d5...	0bf26f677c6c5eda4febe54c6c3f8c7d7cb209a62503bcd111...
student	d3881ede0fac8e2d38a30972d9d8f9f90d3c30f50568346a68...	ab547d2847ce4a4fa3f2374d30d4c1d74fb5bc96e06fc32db2...

Figure 2.21: Contents of the User Table

As explained in Login section, the password and the salt of users will be retrieved from the database. The password that the user used to login will be hashed with ‘Hash 512’ encryption technique twice. The first time is in the client side so the password will not be easy to hack if somebody else receives the AJAX parameters. The second time uses the combination of the first hash and salt to create a password string similar to the one stored in this database (128 characters long).

Currently there are only two accounts available in the system, and the registration page is not provided. The admin and student accounts are used for the admin and student views respectively.

Scenes Table:

+ Options

	title	description	cover	proj1_num_imgs	proj1_files	proj2_num_imgs	proj2_files	proj3_num_imgs	proj3_files	soundtrack
		For report Purpose	sunset-creek-scenery.jpg	3	jpg	1	jpg	1	jpg	Hat Voi Thien Than - Chua ro.mp3

Figure 2.22: Contents of the Scenes Table

This table contains all the information needed for the scene. The title column is used as a primary key in this table. This is the reason for not allowing the title to have special characters or be duplicated when adding a new scene, as well as the read-only property of the title field when editing a scene.

The proj1_num_imgs, proj2_num_imgs, and proj3_num_imgs columns contain the number of files in each projector’s folder. It is mostly used for creating a slideshow or static images.

The proj1_files, proj2_files, and proj3_files columns contain the type of files dedicated to each projector. If the files are images, only the extension will be stored since the files' names are changed dynamically. In contrast, if the file is a video file (only 1 file is allowed in each projector); this field will contain the name of the video file.

2.5.2 IR Control System

The implementation of this system is summarized in the pseudo code below:

1. Continually poll for a key press (this is blocking).
2. Check if the elapsed time between previous key presses is greater than the minimum time delay. If not, return to step 1.
3. Check if the key press matches the valid key. If not, return to step 1.
4. Construct an HTTP request string based on the key pressed.
5. Send the HTTP request to the web server.
6. Wait for the HTTP response from the web server.
7. Return to step 1.

During implementation, some difficulties along the way were noted for this report. The kernel dictates all packets being sent and received by the Operating System (OS). This means that the send() and recv() functions from the network library do not always send the number of bytes required to be sent in one go. These functions needed to be looped until the entire message has been sent. Another problem that arose was that the TCP connection would time out very quickly. This was fixed by caching the addrinfo structure in order to recreate, and connect to a new socket every time an HTTP request needed to be sent.

On the server side, a php script will check for HTTP post requests with valid parameters and control the projector system according to the parameter passed to it. The php script will return HTTP status code '200' for success and '404' if the parameters are invalid.

2.5.3 Streaming Control

For implementation, the file streaming, transcoding, and assembly was taken care of by ffmpeg, the RTP stream was handled by VLC, and the stream reading and decoding on the client Raspberry Pi's was handled by VLC as well. All the Stream Control code was written in PHP.

Each of the MediaStream implementations spawns ffmpeg as a subprocess with parameters that open a file or set of files in the manner needed by the scene. ffmpeg is a multimedia framework that can decode, encode, transcode, mux, demux, stream, filter and play almost any media format currently in use. For audio, it simply reads the file; for slideshows, it aggregates the images in a folder into a H264 video slideshow stream; and for video, it converts the video into H264 if necessary. ffmpeg was the logical choice for this function because of its flexibility.

The RTPStream class is implemented by taking the stream output of any MediaStream instance and passing it to VLC over standard input. When an RTPStream is created it spawns VLC as a subprocess. The VLC instance is passed parameters for it to set up and maintain an RTP stream in multicast. This way, if the stream being played is needed by multiple clients, the stream does not need to be created multiple times. Once the stream is established, a Stream Description Protocol (SDP) file is created which completely describes the stream, and when read by a stream the client allows them to immediately begin decoding the stream. This SDP file is saved in an accessible location by the Raspberry Pi clients over HTTP.

Once the required MediaStream, and RTPStream are setup, a RemoteControl class is constructed to instruct the relevant Raspberry Pi's to play the stream. The *enqueue* method of the class is called, and the URL of the SDP file for the stream to play is passed to the remote Raspberry Pi. Once the stream is queued to play, the *play* method is called which instructs the remote Raspberry Pi to begin streaming.

The StreamConductor class orchestrates this creation of MediaStreams, RTPStreams, and instructs all of the client Raspberry Pi's to begin streaming through a RemoteControl instance for

each client Raspberry Pi. It accepts a scene description object which describes which Raspberry Pi clients are being used and what files should be streamed to which clients.

On the client Raspberry Pi's, VLC was started with the remote control (RC) interface to allow for control by HTTP. This is what is used by the RemoteControl class in the StreamControl package to control the client Raspberry Pi's.

2.6 Testing and/or Evaluation

2.6.1 Testing the Web Interface

The web interface was tested manually by extensively attempting all of its intended functions. First, it was tested for signing in and signing out through two users: the admin user, and the client/student user. Second, it was tested for adding new scenes by adding five different ones. Third, it was tested for deleting scenes by deleting a couple of the scenes that were added. Forth, it was tested for editing scenes by changing and/or modifying the contents of some of the scenes. Lastly, the view scenes aspect of the web interface was tested extensively by logging in to the website after every action was done, and checking that any changes were applied.

Besides manual testing, automated test cases were also created to speed up the testing process. The test cases are written using a combination of Selenium and JUnit. The reason is that Selenium can record and replay humans' interactions with the web interface. By navigating, accessing the webpage using Selenium, and verifying the URLs' returned contents and results using JUnit; the test cases can be executed quickly with high accuracy. The down side of this technique is that Selenium cannot emulate actions on an Operating System (OS). The add scenes and edit scenes forms use uploaders to create and edit scenes, and the file select windows belong to the OS; therefore, Selenium cannot select the files in these windows to upload. Because of this, manual testing was needed. The automated test cases are as follows:

1. Log into the website with student account → expected result: access granted, redirected to "client.html".

2. Log into the website with admin account → expected result: access granted, redirected to “admin.html”.
3. Log into the website with a fake account → expected result: access denied.
4. Log into the website as student and attempt to access admin.html, “addScenes.html”, “editScenes.html” and/or “deleteScenes.html” → expected result: get redirected to “client.html”.
5. Log into the website as admin and attempt to access all pages on the client and admin views → expected result: access granted.
6. Student clicks on the ‘Play Scene’ buttons in “client.html” → expected result: scene is played.
7. Admin clicks on the ‘Play Scene’ buttons in “admin.html” → expected result: scene is played.
8. Student clicks on the ‘Sign out’ button → expected result: redirected to “index.html”.
9. Admin clicks on the ‘Sign out’ button → expected result: redirected to “index.html”.
10. Admin opens the Add Scene form and clicks on the ‘Add Scene’ button → expected result: form is opened correctly and a warning box appears when ‘Add Scene’ button is clicked.
11. Admin opens the Edit Scene page, selects any scene, and clicks the ‘Edit Selected Scene’ button → expected result: form is opened correctly.
12. Admin opens the Delete Scene page, selects any scene, and clicks the ‘Delete Selected Scene’ button → expected result: the scene is deleted from the page.

2.6.2 Testing the IR System

The IR system has been extensively tested manually by pressing all the keys that have function handlers attached to them, at different distances from the IR receiver. In our testing, we found the distance of the IR remote control from the IR sensor to be rather short, but there is no obvious workaround to this except for purchasing a better IR receiver.

We have yet to test very long execution of this program. Proper testing of this will takes months and months to do. We made sure that there are no memory leaks in the code in an effort to have minimized the risk of crashing due to long term execution.

Testing of the server side php script was done by inserting the latest pressed key's enumerated ID into a table on the database, and comparing it to the key that was pressed on the remote. We did not run into any problems testing the code on the server side.

Due to the nature of the C programming language, it is hard to write proper unit tests because of the lack of classes. Instead, the code is asserted many times throughout execution, and will output any error to the stderr before returning a non 0 value to indicate that there has been an error.

2.6.3 Testing Streaming Control

Each class was tested manually to determine if it would function as required. For the MediaStream functions, test scripts were used to determine if the ffmpeg sub-processes were producing the correct streaming output. Test pages were used for testing the output of the VLC streams from the server to the client Raspberry Pi's. For the StreamConductor class, integration testing was also done manually following the addition of the StreamConductor to the scene management component of the project. This testing was done by trying as many of the use cases as time allowed before the demonstration.

3. Marketing and Research

For our audio-video study room, most of the similar products out there are somehow different in some way than what we are implementing. Some places have an audio-video room specifically for entertainment. Such as the playrooms that could be found in daycares, or shopping malls where some parents find it helpful to take their kids there to keep them entertained with video to watch and hear sounds while they shop. On the other hand, other products similar to what we are working on are mostly used for the same reason. Many universities such as the University of Calgary used similar productivity of the audio-video room in their library but with different uses of the main idea. For example, they use

LCD screens to show the videos where most of them are small and never been used for anything but to study or learn something in a lecture.

The prices for most of the similar products vary between one and another. In comparison to what we got for this project, the final price of the product used is higher because it is complicated, and hard to find software that puts an audio, visual environment together without using too many parts of hardware. For example, having LCD screens instead of flat project screens is higher in price.

The potential market of this project is useful mainly in the study environment where schools and universities can have this study, novel, room to create calm and attractive space for students. Creating this room can also be used in other areas such as game centers where they can make simulation of the game life to the gamers like Escape Manor game.

The main reason of our project is to afford light weighted software and have the simplest way to create the perfect environment to the consumers.

4. Conclusion

In the future, the server can provide streaming videos and soundtracks through http links. In the add scenes and the edit scenes forms, admins should be able to enter URLs of videos and/or soundtracks. This would require a re-design for these two pages. For example, the Project 1 Customization can have a toggle button which allows the admin to select between uploading a file and using an HTTP URL.

The database would need to be reconfigured to contain the URL, or detect that the contents of Proj1_files is in fact an URL. Furthermore, in the edit scenes page, if the admin decides to switch from using the uploaded file to using an URL, the system must remove the old files so there are no mismatched contents between the file system and the database.

Due to time constraints, the IR system could not be implemented and tested to the fullest extent. There are some things that must be fixed or could be added moving forward. The recv() function from the network library is a blocking call. This is a problem because the server never responds to the HTTP requests. The problem does not arise when the website is run locally, and the Raspberry Pi has a direct Ethernet connection

to the web server that the website is running on. If the website migrates to an external web server, the latency can be a real problem, causing the program to block for a noticeable longer time period. This blockage will make it seem as if button presses are not responsive. A workaround for the future is to use a non-blocking equivalent to `recv()` or set the socket to non-blocking mode using the `fcntl()` function. With a non-blocking socket, we can create a `MAX_WAIT_RESPONSE` constant and decide what to do if the response is not received within that time.

As of now, all of the keys detected and the functions that trigger are hard coded. If we were to continue to develop this system, we may create a graphical user interface (GUI) to allow the admin to attach functions to different keys on the remote control. For the scope of this project, this feature was emitted.

Currently, the server side processing of the HTTP request does not require an encrypted username and password in the body of the request. This leaves the php script open to attack if somebody has the knowledge and indecency to do so. Leaving this out was primarily done to save time on the implementation. However, if security is a major concern, we can encrypt our HTTP request the same way that we have encrypted access to the administrator account on the website: by encrypting the login and password before sending them over the network, hashing the encrypted password with a unique salt (done on the server side), and comparing the result with the encrypted password stored in the database. The server side code has already been written for this as seen in the “login.php” file, and this feature can be fairly quickly integrated in the IR system.

Future work on the streaming control portion of the project could be in many areas. A custom client software could be created for the Raspberry Pi's which would improve streaming efficiency by taking advantage of the Raspberry Pi's hardware H264 decoder, and could improve the display of static images by simply displaying the image instead of constructing a video stream from static images. Also, synchronization of the audio-video could be looked into. A PHP extension to store persistent data could be constructed instead of relying on shared memory. Startup scripts for the Raspberry Pi clients could be made to run automatically so that VLC does not need to be started manually on the clients.

APPENDIX A – IR System User Manual

Installation

In order to set up the IR system for use on the Raspberry Pi, the following must be done:

1. Install LIRC:

```
sudo apt-get install lirc liblircclient-dev
```

2. Put these two lines at the end of the /etc/modules file:

```
lirc_dev  
lirc_rpi gpio_in_pin=18
```

3. Change the /etc/lirc/hardware.conf file to this (see <http://www.raspberrypi.org/forums/viewtopic.php?f=45&t=7798&start=100>):

```
#####  
# /etc/lirc/hardware.conf  
#  
# Arguments which will be used when launching lircd  
LIRCD_ARGS="--uinput"  
  
# Don't start lircmd even if there seems to be a good config file  
# START_LIRCMD=false  
  
# Don't start irexec, even if a good config file seems to exist.  
# START_IREXEC=false  
  
# Try to load appropriate kernel modules  
LOAD_MODULES=true  
  
# Run "lircd --driver=help" for a list of supported drivers.  
DRIVER="default"  
# usually /dev/lirc0 is the correct setting for systems using udev  
DEVICE="/dev/lirc0"  
MODULES="lirc_rpi"  
  
# Default configuration files for your hardware if any  
LIRCD_CONF=""  
LIRCMD_CONF=""  
#####
```


4. Stop and start LIRC for it to take effect:

```
sudo /etc/init.d/lirc stop  
sudo /etc/init.d/lirc start
```

5. Copy the lircd.conf file into the /etc/lirc/ directory (create one if it is not there). This configuration file was generated using the irrecord program from the LIRC suite and is specific to the Panasonic RAK-DV965WK remote that we are using for our project, more info can be found at the LIRC website (<http://www.lirc.org>). If a different remote were to be used, a new lircd.conf file must be generated using the previously mentioned irrecord program.
6. Connect the IR sensor as per this image:



7. Restart the raspberry pi for everything to take effect.

Compiling and Running

Compile the code using:

```
gcc -llirc_client test_ir_remote.c
```

Run the code (you may need to run this as a superuser):

```
./a.out
```