DUBLIN INSTITUTE of TECHNOLOGY
Institiúid Teicneolaíochta Bhaile Átha Cliath

Education TV
Final Year Project Report

DT228
BSc in Computer Science

Tomas Belskis
C11477418

Supervised by Paul Doyle

School of Computing
Dublin Institute of Technology

06/04/2016

Education TV

Abstract

The objective of this project is to recreate a teaching environment within a web application, that is accessible and free to use. The project primarily focuses on HTML5 technologies with its native video playback element and open source technologies such was WebRTC and Kurento open source media server to achieve video streaming through the browser without any plugins. This project utilizes cloud technologies in order to achieve scaling for the web application and storage of files and recordings. The project focuses on delivering an environment that is easy to use in order to deliver online classes.

The project was created using WebRTC technology and Kurento media server, using kurento-utils.js as a wrapper that simplifies initialization of WebRTC. The web service is hosted on Ubuntu distribution running Node.js web server that is running on Amazon Web Services.

A web application was developed that provides number of features and ease of use in order to efficiently deliver online classes. This allows user to present his own class to a number of viewer users using a choice of video streams (Cam, Screen share or both), with the ability to record his video session for viewers to watch once the presenter is offline. The communication between users is established through group based text chat. And presentation of notes are provided with the ability to upload files and a pdf renderer that display a choice of uploaded pdf.

Education TV

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

_____

Tomas Belskis

06/04/2016

Education TV

Acknowledgements

I would like to thank my supervisor Paul Doyle for providing his input of opinion on how to improve the project and guidance throughout the development of this project, and my close friends and family for supporting and encouraging me.

1 Table of Contents

Table of Figures

Table of Code Extracts

1 Introduction

1.1 Overview of the project

Video streaming is becoming a growing industry as television and communication is moving into online video streaming. Education is also starting to move towards video streaming as there is number of applications being developed and already had been developed to recreate online teaching environments. These applications tend to rely on legacy technologies as they are built around VoIP framework and

Video streaming is becoming a growing industry as television and communication is moving into online video streaming numerous services provide video streaming such as YouTube, Twitch.tv and many others. Education is also starting to move towards online video streaming as there is number of applications being developed and already had been developed to recreate online teaching environments through applications and web based applications. These applications tend to rely on legacy technologies as they are built around VoIP framework and they require either installation of actual application or some type of add-ons or plug in's in order to get the video streaming aspect to work.

With the use of cloud technologies most current communication application are integrating cloud services and building the existing computer applications as web applications in order to not require any installation. This makes the application extremely usable as it is not depended on computers hardware, as long as the hardware supports the browser the application will work on different hardware and OS. Cloud services allows for application scalability at a low cost entry point and are easy to scale by low cost increase.

My project aims to deliver a rich web application that is cross-platform with ability to scale depending on user demands. This application uses newest web technologies that allow for usability without the use of any extensions or plugins. For video streams it uses all new WebRTC technology that allows for browser to capture and stream webcam footage or screen share without any plug ins, currently there is a requirement to install an extension in order to use a screen sharing functionality this is due to security measures in order to secure and protect the user from malicious users observing client's screens. It aims to deliver an environment where lectures can be delivered in a similar manner to real world classroom.

1.2 Project Objectives

The main objective of this project is to successfully research and develop an application that can be effectively used for education department or by any user that is able to provide some knowledge to some users, this project should be able to provide an efficient environment for users that want to provide education and users that want to learn.
List of objectives:

1. Development of core features required to replicate a teaching environment through a web application. This would include web cam and screen share video streams being streamed to multiple viewer clients. Ability for one of the viewers to become a presenter temporarily which is only initiated by the control.

2. Some for of communication for between the viewing clients and the presenter is required, this can be done in a form of a group chat functionality within all the clients.

3. Ability for the presenter to render slides or documents with the ability to upload types of files that can be downloaded by users.

4. Form of login and registration system for authenticating users for particular user cases.

1.3 Project Challenges

The main challenges of successful implementation of this project would be getting accustomed to new technologies and working with an MCU (Media Control Unit). Certainly a great challenge will be posed due to the time frame that is given for this project and working with new technologies that are still in development which have limited online resources and outdated documentation as the development is for these technologies is rapid and the documentation is not being updated to match the development of technologies.

1. Challenge of learning node.js and other node modules in order to implement them into the application.

2. Implementation of screen share poses a great challenge due to in order of implementing browser only ability to screen share there is a requirement for use of some sort of an extension. This extension has to interact with client side of the application in order to select a particular screen to be shared. This also has to be set up with proper constraints for media stream in order to work with WebRTC.

3. Ability to switch presenters the logic behind this will be difficult to implement due to the limitations of how WebRTC media streams are being set up and the negotiations that are required before the media is streamed. After negotiations there is a requirement for MCU control as to which client is feeding the WebRTC media stream into the MCU. The control to switch presenter will be a real challenge as there is complexity both on client side peer connection and negotiations on server side.

4. Due to core technology for this project being is WebRTC which is only peer to peer and the requirement for my application is the need for n viewer client support, the need for the use of MCU was a requirement. The challenge posed is that there weren't many open source MCU's to make use of that worked with WebRTC due to it being relatively new having limited options I was presented with a MCU that is still in development and having limiting and outdated examples in regards to most of it's implementation.

5. The use of cloud services to host the web application on the cloud, and combining all of the features under same application.

6. Making it scalable for mobile applications and functional to an extent.

1.4 Structure of the document

Structure of the document consists of:

Research,

Education TV

This section focuses on the research that was carried out for the project including alternative solutions that were looked at for guidelines and general specification to carry out a successful implementation of the project.

Design,
This section of the report focuses on the approach of design for the project with project methodologies and prototypes. It describes the decision process when it came to technologies that were used.

Architecture and Development,
This section focuses on the architecture of the application and development cycle, all the components that consist within the application and what worked and didn't throughout the development that I had to make changes from my initial design.

System Validation,
This section of report describes the process that undergone to validate the system, testing methods that were performed and includes the feature demonstration of the project.

Project Plan,
This section of report analysis the project implementation and reviews the changes that where made from initial design.

Conclusion,
This section discusses the learning outcomes achieved throughout the implementation of the project and if it was a successful implementation of an original idea and if the things that wanted to be achieved for this project had been achieved.

2 Research

2.1 Background Research

The background research was sparked by my personal experience as part of a global classroom module which was done online using YouTube streaming and google hangouts. This Global Classroom module consisted of a lectures being presented online through YouTube streaming, we as students had to go and view the livestream that was streamed by our lecturer. This allowed for very limited interaction with the lecturer and live discussion was not possible with the lecturer, in order to communicate with the lecturer, we used hangouts text based group chat. For this module we were also required to do presentations which were quite difficult to perform as it were done online. In order to do live presentations online we used google hangouts screen sharing capabilities, as part of a group each student had a set number of slides to present, this means that there was need of constant switching of who is screen sharing at that particular time. There were constant issues with student machines failing or just in general switching between student's screen share was complicated enough and caused delays in presenting smooth presentation. This sparked a research in developing an environment that simulated an actual classroom where lecturer could present his lectures, there would be possibility for discussions with ease between the lecturer and its students. The solution to this problem would accommodate all of the different features that were used within global classroom in a single environment, this would include group chat between students and lecturer for discussions and student's being able to do presentations with ease all in the same environment and not using multiple technologies.

This online classroom environment would solve not only the problem I was having with global classroom module, but could also solve the problem of future students that can't afford accommodation near their university or commuting to the university. Currently in Ireland there is shortage for student accommodation all over Ireland especially in Dublin, the rent for accommodation is rising at an exceptional rate and public transport prices are rising too on top of increased university fees. This possibly eliminates families that can afford to send their children to university within each year of increased prices in accommodation, travel and university fees. My online class environment would try and solve issue of travel and accommodation, as there would be no need for accommodation or travel, as all the attendance to lectures would be done through the online environment. We can look at the figures of how much money this would save for future students. As per daft (the most commonly used website in Ireland for renting and letting) report for 2014 Q3 to 2015 Q3 average nationwide rent is 963 euro, rent has been raised 9.3% nationwide.[1] There is lack of accommodation for students in Dublin, single shared rooms range from 300 - 600euro per month depending on Dublin area (Jones and Gannon, 2015). This causes serious issues for students to find affordable accommodation within Dublin which could potentially prevent them from attending university. Another cause is the rise in national public transport fees for students that commute to Dublin, I myself commute to Dublin from Co. Meath which comes to 210 euro per month pre rise in transportation fees, and the fees for public transport are being raised in 2015 again. As per NTA announcement public transport fees are set to increase by 5% for Bus Eireann operated services, LUAS set to increase by 60 euro annually, Dublin bus services will see a small increase in its services, Trains as said by Irish times "A maximum increase of 3 per cent on monthly and annual fares has been approved while a 4 per cent increase in the cost of

a three day or seven day travel ticket has also been granted in a move to encourage more people to use cheaper Leap cards. [2] (Pope, 2015)" In conclusion every form of public transport in Ireland is seeing increases in its fees to already expensive service. This causes major issues for families with low incomes to send their children to university, they might be able to afford the initial university fee but accommodation or commuting fees are not affordable for the parents.

By eliminating the need to physically attend lectures with my online classroom environment I open up possibilities for people to study from their home, and this not only concern for families with low income but also students that might have received some injury throughout the year and cannot attend lectures physically or students that have disabilities, physical movement impairments that are also having difficulty attending lectures. This opens up the possibility for a student that currently cannot get a university degree due to their problems of attending university in person possible.

2.2 Alternative solutions

2.2.1 The Big Blue Button

The BigBlueButton is one of the similar solutions to my problem that I'm solving, but it is still in development it's not a complete project. [3] The problem that bigbluebutton is attempting to solve is to develop a tool that can be used for online learning with simplicity and ease of accessibility for students that aren't able to attend lectures due to numerous reasons examples of such reasons could be injuries or financial reasons. The bigbluebutton is an open source project that started in 2007. The high level architecture of the bigbluebutton.

The BigBlueButton is one of the similar solutions to my problem that I'm solving, but it is still in development it's not a complete project.[3] The problem that bigbluebutton is attempting to solve is to develop a tool that can be used for online learning with simplicity and ease of accessibility for students that aren't able to attend lectures due to numerous reasons examples of such reasons could be injuries or financial reasons. The bigbluebutton is an open source project that started in 2007. The high level architecture of the big blue button.

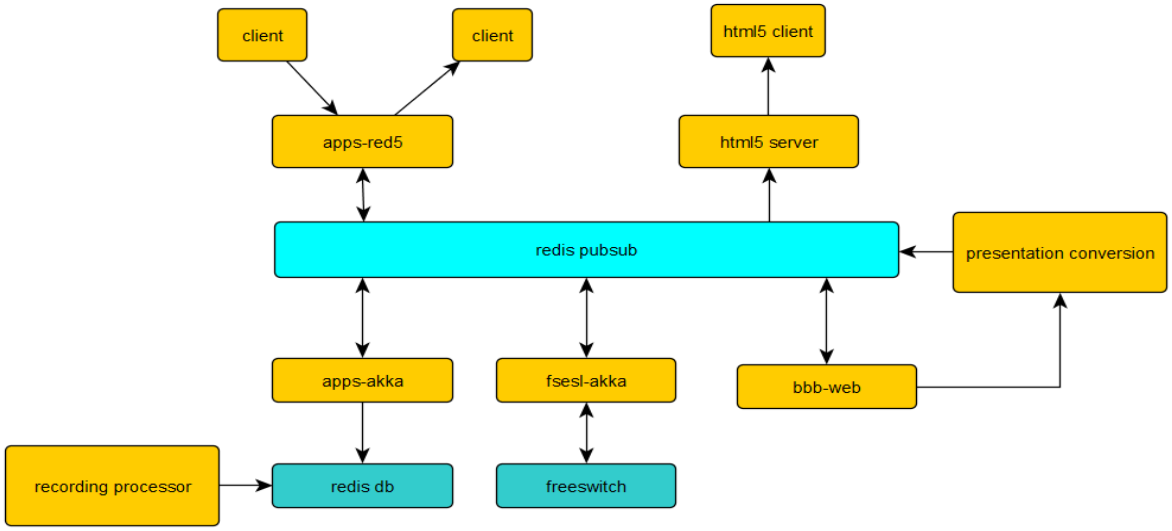


Figure 2-1 Architectural Mode BigBlueButton

Education TV

This high level architecture diagram of the big blue button shows us the client's connection to apps-red5. RED5 is an open source media server that allows connections using RTMP (Real Time Messaging Protocol).[4] It is used for an alternative solution to the flash communications server, red5 was primarily used for streaming flash at the time of bigbluebutton development but as of now it has moved beyond just serving flash streaming and will be incorporating WebRTC in the near future.

"The HTML5 client and server is built using Meteor and communicates with the other components of the system through Redis pub sub.[5]" "Meteor is a full stack JavaScript application platform", it is used to build web applications and mobile apps, it can work in conjunction with frameworks such as angularjs and react.[5] The HTML5 portion for the bigbluebutton is still in development as BigBlueButton began development before HTML5, now it's adapting the technology provided with HTML5. The architecture of HTML5 within bigbluebutton is.

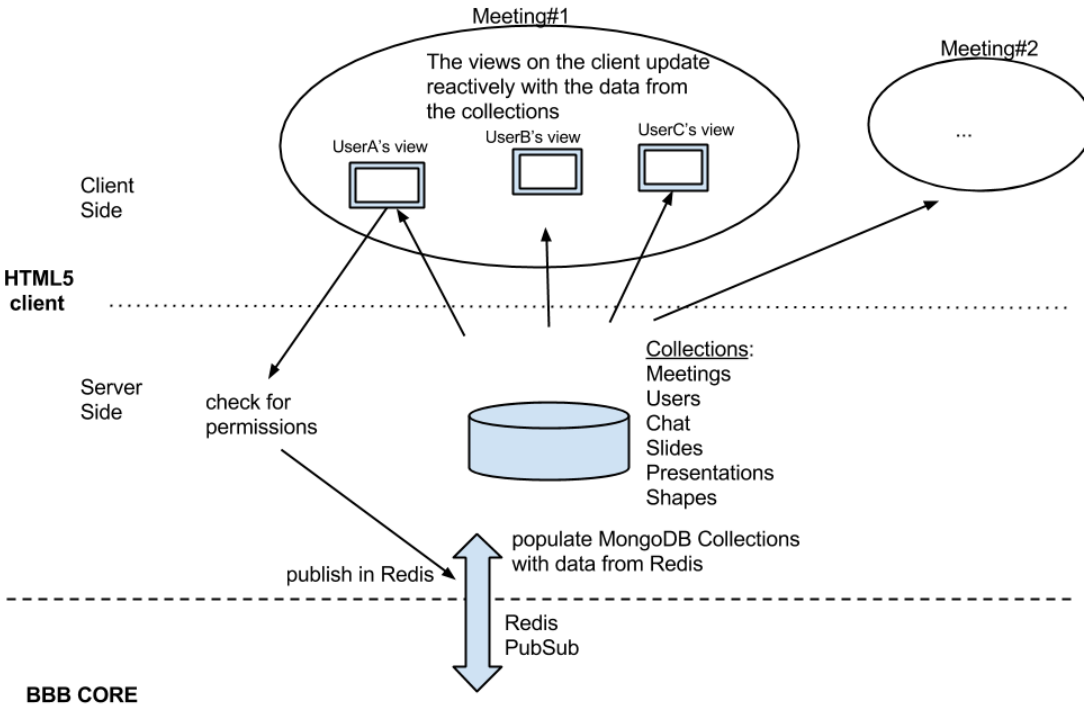


Figure 2-2 High Level Architectural mode BigBlueButton

"The HTML5 client is implemented using Meteor.js in CoffeeScript". [5] The architecture displayed here shows the client's interaction with the collections database which keeps track of meetings, user's, slides, presentations and shapes. Redis PubSub is used to structure data before insertion to a MongoDB collection.

BigBlueButton is a great open source project that is in development and covers a lot of the functionality that my project is going to accommodate. But BigBlueButton is using outdated web technologies like the flash player and does not make use of WebRTC which simplifies a lot of the process in delivering same functionalities as the BigBlueButton.

2.2.2 WebEx

WebEx- is a web based online conferencing and webinar tool, this solution is not exactly what I'm trying to implement but it's similar.[6] It is not open source solution but it has some free features but they are very limited and most of the other features are paid. WebEx is quite well known conferencing and webinar tool as it has a high following and many users. It is developed by cisco multinational company that works on different technologies and WebEx is their VoIP project.

In respect to WebEx architecture, "*Cisco WebEx Social is a collaboration platform that allows you to work in an environment in which you can easily share information such as documents, videos, and presentations, conduct meetings, click-to-dial a contact, post information, join communities, participate in discussion forums, create blogs, and much more*.[7]" Cisco WebEx incorporates multiple technologies developed by Cisco industry and additional technologies developed by other companies to provide a lot of the services to host conferences and webinars, examples of such services would be Cisco show and share, WebEx meeting center etc.

Figure 2-3 WebEx High Level Architectural model

This architecture diagram shows us multiple client support and support for a lot of different types of clients. Cisco has developed its own WebEx Social Application Framework that sits on top of virtual data layer.[8] This framework provides application services that are common across system features.

Education TV

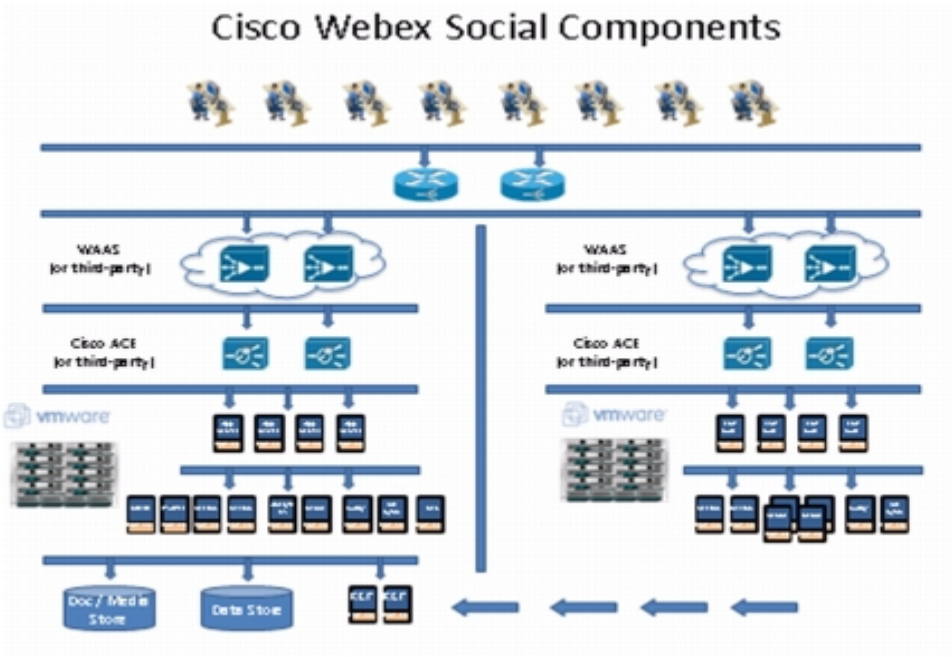


Figure 2-4 WebEx lower tier architectural model social components

Cisco WebEx is still making use of outdated technologies for it's conferencing application. It is making use of XMPP protocol. Uses multiple data stores internal and external, collaboration data is maintained within oracle and Mongo databases. WebEx uses a node based architecture where each node connects to central provisioning server and then downloads specific role configuration.

In conclusion WebEX doesn't make use of new web technologies that are available today, it it's using outdated architecture in delivering conferencing application and therefore it's falling behind other conferencing applications that are available, due to compatibility issues and limited platform support. WebEX- is a web based online conferencing and webinar tool, this solution is not exactly what I'm trying to implement but it's similar.[6] It is not open source solution but it has some free features but they are very limited and most of the other features are paid. WebEx is quite well known conferencing and webinar tool as it has a high following and many users. It is developed by cisco multinational company that works on different technologies and WebEx is their VoIP project.

2.2.3 Google Hangouts

Google Hangouts - is a conference web application developed by google. It provides the ability for group video calls and has group text based chat implementation.[9] Google Hangouts main draw is that it does not require any installation and is ran easily with the use of a web browser. Google hangouts also has implemented VoIP for it's service which allows for calls to landline and mobile phones. Google hangouts has wide range of device support due to it's android, iOS and desktop applications. Google hangouts had used the XMPP protocol but once google terminated the use of Google Talk it

ceased the use of XMPP protocol.[10] Google Hangouts is conferencing simplified; it doesn't require additional plug ins except if it were to used on Internet Explorer.[11]

In order to offload most of video and signaling traffic other video communication applications such as skype or chat roulette make use of peer to peer connections. But According to google real time communications tech lead Justin Uberti "To support hangouts, we built an all new standards based cloud video conferencing platform.[12]" He also added in that google hangouts uses a client server model that leverages the power of Google's infrastructure.

Google after terminating XMPP for hangouts and terminating google talk, showed a lot of interest in WebRTC and backed up it's development in 2011 they released WebRTC as open source project for browser based real time communication. But google hangouts didn't make the switch to use of WebRTC right away as it was early technology. According to an article that reverse engineered Google hangouts, hangouts gained support of WebRTC with chrome version 36. But due to limited documentation on how Hangouts is implemented it's really hard to determine google hangouts architecture.

2.2.4 Skype

Skype - is peer to peer VoIP client application developed by KaZaa in 2003. Skype offers numerous features such as instant messaging, group video calls, screen share, peer to peer file share and landline, mobile phone calls.[13]

Skype architecture consists of nodes which are ordinary hosts and super nodes, such hosts for example an ordinary host would be skype application that can voice call and send messages.[14] A super node is an ordinary hosts end point with a public IP address having sufficient CPU, memory and network bandwidth is capable of becoming a super node. Ordinary host connect through super node and in successive connection they must connect to skypes login server for authentication. Apart from login server skype doesn't make use of any other type of server. Nat and firewall traversals are important for skype, as the enable connections amongst clients. Each skypes node uses a variant of stun protocol in order to determine firewall type on client's machines and NAT traversals. Skype network is an overlay network it contains a host cache that holds information such as host IP address and port numbers of super hosts. Skype uses TCP protocol for server signaling and TCP and UDP for media transfer just as WebRTC does. Skype client listens on set ports for incoming calls or messages maintains a list of other nodes within host cache. All of the call handling is done through super node hosts. The end user has no control over which super node it connects to or which node is determined to be a super node as it's chosen based on resources available.[15]

Skype used to be an application only support which required an installation on your computer. Currently skype is implementing a browser based web application that can be used just like the skype desktop application. Skype web application uses WebRTC technology in order to deliver same features as the desktop application. Currently skype web application requires a plug in but it's due to still being in beta once it fully incorporates WebRTC it will not require any use of plug ins to have complete functionality of Skype desktop application within a browser.

Education TV

In conclusion skype is a powerful VoIP client with multiple platform support ranging from desktops to mobiles and soon it will expand to having to a complete web application. Skype has stayed around for many years and has proven that constantly evolving with each new technology that is being released it manages to stay in lead as a VoIP client application for video calls and instant messaging. Where many other similar applications have failed, like MSN messenger which is part of skype due to Microsoft buying skype.

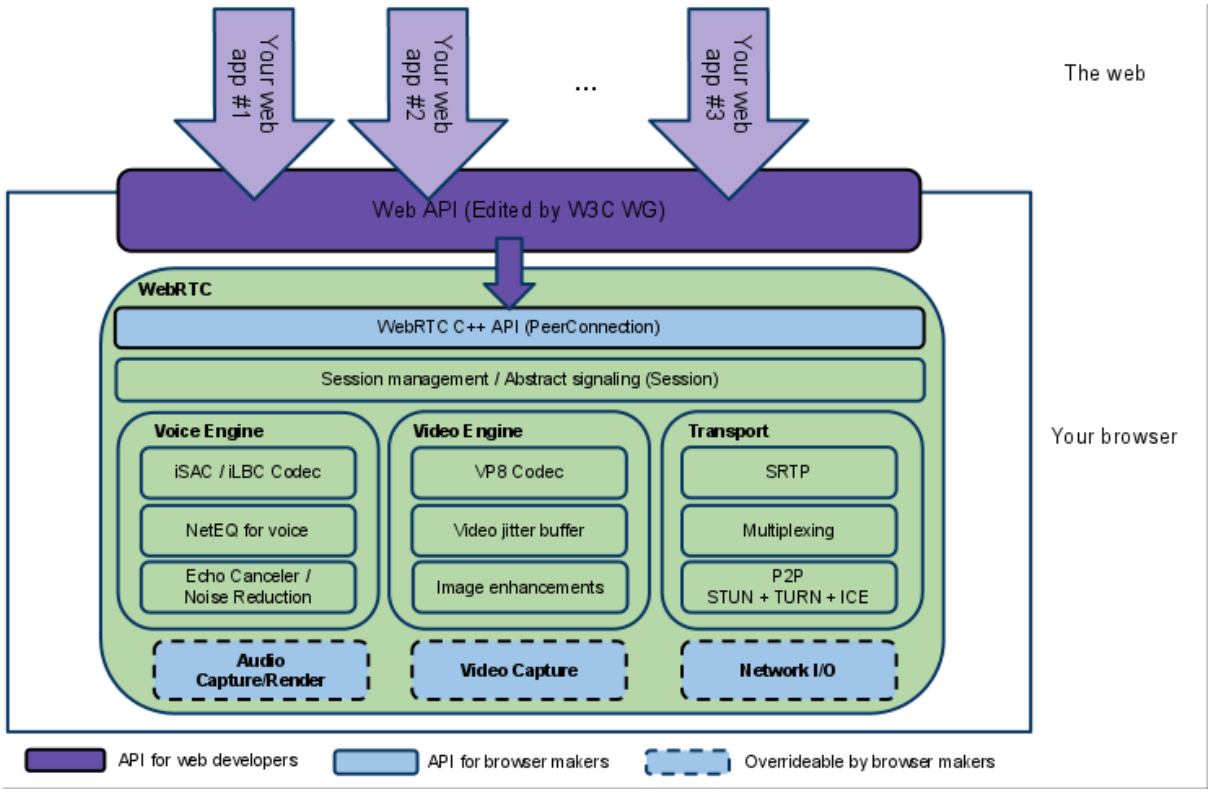
2.3 Technologies

The technologies that I focused my research on are primarily the most common used and newest web application technologies, as my project is a web application. Most of these technologies are fairly new so I didn't have much experience before research with these technologies. In order to familiarize myself more with these technologies I had looked at tutorials of some projects which provided me with some understanding of how these are put in and used in an actual working example. Also I looked at projects that are fully fledged out and are using these technologies to see what can be achieved with the use of these web technologies.

2.3.1 WebRTC

WebRTC is Video call, web chat and p2p file sharing without the need of either internal or external plugins to be implemented with the use of WebRTC (Web Real-Time Communication) API, this is drafted by World Wide Web Consortium.[16] WebRTC enables all kinds of real time communications. the main draw to use WebRTC is because its real time communication without the need for additional plug-ins. Which adds accessibility for wider range of platforms.

The Architecture of WebRTC consists of multiple layers.



Education TV

My web application would go on top of the WebRTC API layer. WebRTC provides peer to peer connection, this supports a client browser to another client browser communication.[17] This model does not work for my project due to my project requiring multiple clients connected to single broadcasting client. There are workarounds that use the peer to peer model by creating multiple sessions to accommodate multiple clients through peer to peer connection. Or just having all the clients interlinked with the use of peer to peer. But in my implementation I will make use of a media server to transfer media traffic to multiple clients which acts a media control unit and redirects media stream.

I will use WebRTC to implement the following features of my application, Webcam Stream, Screen Share, Audio Stream and file sharing. All of these features are part of WebRTC API.

2.3.2 HTML 5

HTML5 - is an updated version of HTML that has recently been introduced, it delivers a lot of new powerful tools with this update that makes my project possible.[18] With this html revision we get to use a number of new and powerful tags that weren't available before and you would of had to create workarounds to in order to execute some of the functionality that is easily performed with HTML5 and their new element tags, without html5 revision this project would not be possible as HTML5 pushes ease of implementing large scalable web applications primarily focusing on client side functionality and less to deal with the server.[19] HTML5 is developed by and revised by W3Schools. I will incorporate HTML5 in my application in order to make use of the HTML5 <video> tag, this allows in conjunction with WebRTC to play video stream in a browser without any need for plugins or extensions.

2.3.3 NodeJS

Node.js - is an asynchronous event driven framework, it is an open source technology for developing server side web applications.[20] Node.js works with the scripting language JavaScript, Node.js has been developed by Node.js Developers Joyent and GIT Hub Contributors as this is an open source technology. Node.js uses Google's V8 JavaScript engine in order to execute code it contains a built in library to allow applications to act as a web server. Node.js can be in particular be used in conjunction with other frameworks in order to accelerate web application developments, such other frameworks would be Express.js, Socket.io and Connect. Also Node.js has support for Coffee script which is an alternative form a JavaScript. Node.js is primarily used to build web servers which is what i will be using it for within my project, as Node.js works well in conjunction with WebRTC. Node.js provides us from deadlocking processes there aren't any locks. Considering no function in Node.js directly performs I/O, so the process never blocks, this allows for us to develop scalable systems without worries of process deadlocking. Node.js designed without threads, but you can still take advantage of multiple cores in the environment by spawning child processes it allows for sharing sockets between processes.

2.3.4 Mean Stack

Mean Stack- web technology stack used as architecture for developing web applications.[21] It makes use of technologies such as MongoDB (NoSQL database), Express which is minimal node.js web application framework, Angular extends HTML vocabulary for application development and node.js for networking side of application development.[21] Mean stack use seemed at first tempting but the

only technology from this task that applies to my project the most would be node.js, other technologies part of the mean stack doesn't apply to as much to my project development.

2.3.5 Lamp Stack

LAMP Stack- another web technology stack used as architecture for developing web applications.[22] Lamp stack consists of Linux, apache, MySQL, PHP/Perl/Python. This stack is not as compatible with WebRTC due to the fact that it's not supporting node.js. But reviewing both stacks I decided against using either of them due to the fact that I do not need to make use of all of the technologies served by both of the stacks.

2.3.6 Amazon AWS

Amazon AWS - Amazon provides low cost infrastructure platforms in the cloud.[23] There are number of services that provided by Amazon AWS but I focused on Amazon Elastic Compute Cloud (Amazon EC2) which is a web services that provides scalable compute power in the cloud. This service allows to create virtual servers in the cloud for use of deploying your web applications or web services. The user has to manage the virtual server himself as this Infrastructure as a service(IaaS) cloud based service model. The user has complete control of the instances of servers he has deployed on the cloud. This service also works in conjunction with other services provided with Amazon AWS.

Another service provided by Amazon is the Elastic Beanstalk which is Platform as a service (PaaS) cloud based service model.[24] This allows the user to deploy his web applications depending on which platform he chooses there are multiple platforms offered by Amazon AWS. This service handles capacity provisioning, load balancing, scaling, and application health monitoring. All the user has to do is just deploy his web application through the provided service and everything is taken care of by Elastic Beanstalk. This is really attractive service to use as it's really easy to quickly deploy your application and I plan on using this service for testing my web application, due to how easy it is to deploy your web application.

2.3.7 Kurento

Kurento - is a WebRTC media server that improves the multiple capabilities of WebRTC in order to improve video stream and broadcasting with the use of WebRTC.[25] WebRTC is a peer to peer connection, but with the use of Kurento, we can stream users getUserMedia to a Kurento media server and from Kurento media server we can stream the media to multiple clients and different types of devices computers, phones and tablets.

Kurento solves the problem of the issue of WebRTC having peer to peer media transfers, because for my project I am implementing multiple client support, peer to peer connections provide with WebRTC aren't sufficient I need to a media server in order to stream media to multiple clients.[26] Kurento has multiple implementations it can either have a direct connection from WebRTC or with the use of NodeJS server, this another reason why I will be using Kurento as my chosen media server in the

works with WebRTC is due to the fact that it works in conjunction with NodeJS which I had already decided to use as my server signaling server for WebRTC.

2.3.8 Red 5

Red5 - is an open source media streaming server implemented in java.[27] It is free software with a paid version Red5 Pro. Currently Red5 supports only flash send and receive video, which is not an attractive service to my solution due to my solution using WebRTC with HTML5 element which as of currently Red5 has no support for and plans to incorporate the support in near future. The benefit of using Red5 media streaming server would be due to the numerous powerful API's it provides for managing your media stream.[28] But due to poor documentation and no support for WebRTC and HTML5 I decided against using Red5 as my media streaming server.

2.3.9 WebSockets

WebSockets - "The HTML5 WebSockets specification defines an API that enables web pages to use the WebSockets protocol for two-way communication with a remote host. It introduces the WebSocket interface and defines a full-duplex communication channel that operates through a single socket over the Web.[29]"

Web sockets API is standardized by W3C, which is also working on WebRTC and HTML5. Web sockets is a protocol that sits on top of TCP protocol, this improves the interaction between the browser and the website.[29] Web socket is required when using WebRTC for signaling the server and finding a peer. Web socket implementation will be used in conjunction with NodeJS server for signaling.

2.3.10 Meteor

Meteor- Meteor is open source web application framework written in JavaScript.[30] It provides the use of many libraries. It supports Node.js as it's server and MongoDB as a database. Meteor is great for rapid application development this allows for Meteor developers to quickly build scalable functional prototypes that can be implemented as full scale applications. Meteor is nice piece of web application technology, but I do not require it for my project.

2.3.11 Angular.js

Angular.js- is another web application framework written in JavaScript and maintained by google.[31] Angular.js extends the capabilities of HTML by reading HTML page which has embedded into it additional custom element attributes angular interprets these attributes binds as inputs or outputs and applies it to a JavaScript model. Angular provides for framework only for client side development it focuses mainly model view controller and model view view model architectures along with components commonly used in rich web applications. Angular.js is used for improving the presentation of your website, I don't find the need to use Angular.js for my solution due to my UI being simplistic as possible and can be done with the use of basic HTML. If I feel like my UI is lackluster in further development, I will reconsider implementing AngularJS into my web application.

2.3.12 React

React- Is an open source library used to provide the view for data rendered as HTML.[32] React is also based on improving the visualization of data presented on your HTML page. It allows for use of custom HTML tags due to React library, to present data in a way that is not possible just with HTML.

Education TV

It emphasizes on data display and separation of components on single page designs. This is not applicable to my project and I will not make use of React.js libraries.

2.3.13 Coffee Script

CoffeeScript - is a lightweight language that compiles into JavaScript, it's purpose is to simplify JavaScript and extract the best qualities of JavaScript in order for simplified development of JavaScript like applications.[33] I might make use of CoffeeScript during the development of the project due to the fact that it improves the readability of JavaScript and it easily trans compiles into JavaScript. CoffeeScript would be only used in order to speed up the development process due to it's clear ability and simplicity of use.

2.4 Additional Research

2.4.1 Teaching and Learning

I performed research on the current day methods of teaching and learning because my project is based on producing a learning environment through web application I looked into researching how is education delivered today in schools with the current day technology.[34] Currently primary schools have incorporated the use of Tablets and IPad's, in the presentation of teaching. These iPads are used for having your school books and homework tasks. This reduces the weight from students bringing school books in everyday and possibly forgetting to bring in their books as a tablet is very thin and easy to carry. Having this technology at teacher's disposal allows teachers to introduce different teaching methods dues to tablet supporting many applications that can be used for education. Tablets aren't the only form of technology teachers make use of today, projectors are another form which allow teachers to display presentations created on computers.

This influences students positively as it improves their learning.[35] Learning is improved due to active learning imposed with the use of tablets, there are numerous teaching applications that the teacher has at their disposal to make use of in order to stimulate learning. There is a large eBook library to be used by students. The only missing feature would be the presentation of a class on the iPad itself. This would be what my attempt at implementing the online classroom would allow me to do it wouldn't be much of use for students that do attend the class physically, but it would be useful for students that are unable to attend the class physically.

2.4.2 VoIP (Voice over IP protocol)

"VoIP involves sending voice transmissions as data packets using the Internet Protocol (IP), whereby the user's voice is converted into a digital signal, compressed, and broken down into a series of packets[36]." After these packets are formed it's send over the network just like any other packet would be email or a webpage. This allows for communication computer to computer, computer to landline, computer to mobile and any phone that would have a VoIP application installed.[37] The current phone system is inefficient and expensive to maintain therefore the calls current phone system is expensive, the reason for the greater cost of phone system is due to the way it's implemented using circuit switching technology. Current day providers are starting to make use of VoIP technology to route their phone calls into an IP gateway, this allows for reduction of overall bandwidth due to the fact that the it uses packet switching rather than circuit switching. But this is a slow process of implementation for current phone service providers therefore it will take some time, but VoIP is the

future in voice communication due to economics and infrastructure reasons. The reason why VoIP is so attractive is because broadband connectivity support is growing around the world, due to this you can make phone calls with the use VoIP anywhere you have internet connectivity. Another reason why VoIP is better than current phone system is that it's much cheaper to maintain, therefore the service you are acquiring for your money is much better, there is no such a fee as roaming charge for example.

Viber is a free phone application that allows for VoIP calls to any other of the application users. The calls are completely free of charge due to VoIP technology, the calls cannot be made to landlines or mobiles but that is a paid service and there are other applications that provide calls to landlines and mobiles with a small charge example would be skype.

This is relevant research to my problem in the way that WebRTC simplifies the communication further. Implementation of VoIP is expensive. WebRTC eliminates a lot of problems that invoke the costs of VoIP, WebRTC enables for developers to make use of browser built in functionality to stream not only audio but video too, without any additional software.

2.4.3 Security

Security is a big concern with ability of to remotely stream another client's webcam or screencast their desktop just due to how powerful of tool WebRTC is.[38] Google has a lot of concerns with the user of WebRTC and even though they highly support it and encourage development of applications with the use of WebRTC, they are adding numerous restrictions with each new release of Chrome. The reason why WebRTC poses big risks to user's security is due to the way WebRTC works, every chrome browsers supports the getUserMedia API call from WebRTC, this allows the browser to capture video feed from a number of camera devices on the user's systems. There can be website that has a script that makes use of this to redirect user's camera stream to a server and potentially have a live feedback running of the client's camera consistently without the user realizing. WebRTC technology is quite new therefore there hadn't been many concerns with the security of it there haven't been put up many restrictions. Now WebRTC imposes big security risks through invasion of privacy and attackers could bug out the system and impose greater damage even perform denial of service attacks. The user has to be notified who is instantiating a call, what resources are requested (microphone, video cam) and where the footage is being streamed to exactly. Media can be received only with the consent of the client; client must be always notified this would protect the user from malicious sites capturing video from the webcam or audio. As the user would have the ability preventing unknown sources from capturing its video and audio this adds a small layer of security to user's privacy.

There is another threat in addition to camera and audio sharing, WebRTC allows for screen sharing functionality.[39] This poses risks such as over sharing, the user might not realize that he is sharing some private information due to this, the user might think he is sharing his current view, but in fact a view of an application that is running on the computer is being shared too. The attacker website can request the user to screen share a certain window, but in fact capture a different window perhaps a tab with user online banking information. The security experts suggest that the security around screen sharing should be much stricter than the video cam and microphone permissions request. As currently in chrome you only need to give permission the use of webcam and microphone for a certain website

once and it saves this setting for that website, this means that the user upon entering the website again wouldn't get requested for user's permission again and the website would already have access to the camera and microphone. If the same permission request would incur with the use of screen sharing, this would allow the website to capture user's screen without notifying the user, user being unaware might display some vital information on his screen and attacker might make use of this information to further cause damage.

As of now google is adding extra requirements in order to enable screen sharing.[40] In early versions of screen sharing with WebRTC there was only a requirement of an enabled flag within chrome in order to give permission of screen sharing, there were no other requirements. But google has seen that this is a huge security risk and has moved the flag only to the developer's version of chrome. On ordinary chrome they have implemented the requirement for an extension in order to give permissions for screen share, this way the user by adding the extension to his chrome version knows exactly which website has the permission to screen share and another website cannot make use of the extension to allow screen sharing. Also in order to allow for screen sharing with WebRTC you are required for a secure server. The server has to use HTTPS and SSL certificates in order for screen sharing to work this adds another layer of security. As it restricts attackers access to the server and makes it difficult to acquire the screen share stream.

This affects my project in the way that I will have to configure my server to use HTTPS as screen sharing, video and audio streams won't be possible due to the new chrome security policy that is being implemented this year, this adds a bit of difficulty in configuring the server as previously SSL, TLS certificates have been all paid but now alternative free sources are appearing that are providing these certificates for free.[41]

2.4.4 Streaming

Streaming media is multimedia that is constantly received by a user while it is being delivered by a broadcasting client.[42] Streaming media came about when downloading media files was becoming impactful on the user's machine due to limited hard drive space on the machine and media files were quickly increasing in size. This created the need for implementation of a new solution which became streaming media this is a solution that allows you to watch the media that is being downloaded on the fly, instead as previously it was only downloadable which would require the users to download all of the media before being able to see it. Early development of streaming media involved in pre-buffering a certain part of the file and then streaming that part to clients, there were no rewind, or fast forward functionalities. Constant buffering was causing frustration to clients, therefore new streaming media protocols were in demand. Servers and protocols dedicated to streaming were developed to improve overall quality in streaming media, the first of such streaming servers was RealAudio server and a RealAudio player was used to receive the media stream. RealAudio player used TCP and UDP transmission layer protocols these protocols allow for transmission of media in bulk of compressed audio packets. Due to the way transmission through UDP protocol works, packets were being lost when streaming media, causing streamed media quality to suffer and missing parts of the streamed media couldn't be recovered as the packets are lost during transmission. This caused the development of RealAudio codec that made sure that all the packets arrived in sequence. This meant that

compressing parts of the media that was going to be streamed over a transportation layer, and having it be transmitted at a fixed bit rate to prevent bit loss during transmission.

Overall today's media distribution consists of numerous changes as the transmission of media goes through a number of intermediate servers before it reaches the client. There is still work being done on internet's infrastructure to improve the bandwidth of streaming and quality of streaming produced. Current day media distribution is not only possible from a file; live media distribution is also available. Where by encoding the encoding engine captures and digitizes the analog video compressing it and transmitting over the network to the server, the server redirects this media stream to non broadcasting clients. Alternatively, there is a way for the server to receive pre encoded data with the use of Simulated live transfer agent this allows for reading of pre encoded data and transfers just the same way as it were if it was encoded. The server is responsible for delivery of data to all corresponding connected clients, simply by unicasting the coded video media to each client individually using one-way data streams.

Live Streaming media has become a has seen a large usage jump in recent years and its popularity.[43] There are numerous streaming platforms available currently, popularity of such platform provider depends on the region, Asian region usually have their own streaming platforms that are usually more popular from western regions streaming platforms. Most popular streaming platform in western region is well know TwitchTV, this streaming platform allows for a certain broadcaster to create his own personal channel, through which with the use of broadcasting software he can broadcast his running applications and screen share also broadcast his webcam. Twitch is currently really popular and growing streaming platform currently mostly used for video game streaming. Twitch currently requires an application that does the capturing of the webcam and screen sharing and forwards this media stream to the channel setup on TwitchTV platform. Naturally TwitchTV streaming was only available for desktop computers, but now it has seen support on consoles. There are mobile phone applications that allow to view the streams but not stream as of now. TwitchTV uses Real time messaging protocol over HTTP tunneling for it's servers to transfer media from broadcaster to the server and the original streaming is being transferred through HTTP live streaming from streaming servers to viewing clients.

YouTube is one of the biggest media streaming websites that exists in current day, YouTube recently had also implemented a live streaming service, this service might not be as popular as other live streaming platform providers but nonetheless it's a live streaming platform.[44] YouTube uses HTTPs live streaming as the other platforms. Due to recent advancements in HTML with HTML5 YouTube now supports HTML5 video streaming which is a huge advancement over previous types of video streaming. YouTube still makes use of HTTPs live streaming protocol. HTML5 video playback is currently is the default player, but due to old browser versions not supporting HTML5, flash is still used when browser does not support HTML5 as a fallback technology. HTTPs livestreaming protocol works in the way that all the media needs to be transferred as a HTTPs objects. These objects can be cached by proxy server and then they can be transferred and distributed like any other HTTP object like images or html websites.

2.5 Resultant findings and Requirements

Education TV

2.5.1 List of requirements

The list of requirements was finalized after the completion of research, this list contains all of the required features and usability functions for this project in order to deliver an education rich environment for being able to successfully provided a system that can be used to deliver a form of classes online.

1. Webcam and microphone broadcast to connected clients.
2. Screen sharing to connected clients.
3. Switching to another client for broadcasting.
4. File upload.
5. File rendering.
6. Group chat (text based chat).
7. Notification of connected clients.
8. Login/Registration.
9. Scalable UI for viewing on mobile devices.

2.5.2 Technical Requirements

The list bellow contains all of the technical requirements that have been discovered thought research in order to successfully deliver a system for delivering a successful implementation of initial proposal.

1. Knowledge of WebRTC API, and development process behind it including signaling, peer connection establishment and Session Description Protocol.

2. A node.js webserver, this server is hosted on a cloud based service such as Amazon AWS or Azure. For this project I will be using Amazon AWS EC2 to host my node.js server.

3. Database MySQL database installed on Node.js server to hold records of clients.

4. HTML/CSS knowledge.

5. JavaScript language knowledge.

6. JSON object knowledge, data protocol for transferring messages through the WebSocket.

7. WebSockets, used to transfer message client to server and vice versa. WebSockets are also used for transferring signaling message from between clients, signal the server when requesting peer connections to other clients in order to find out information about client and if their machine supports WebRTC and establish a connection between peers.

8. Media server (Kurento Media server) The use for media server is to incorporate one to many streams as WebRTC is only serving peer to peer connections. There is another solution to apart from

using a media server, which is to use multi sessions for a single broadcaster, what this means is to create a different session for each connected client, so the broadcaster would have peer to peer connection to each client but multiple clients connected to a single broadcaster. This implementation is my fall back solution if the implementation of media server fails.

2.5.3 Research Analysis

2.5.3.1 Webcam and microphone broadcast to connected viewer clients

Will be implemented with the use of WebRTC API, the getUserMedia () function call captures camera and audio. You are able to redirect this stream to either a peer connection or a media server, for my implementation I will be directing it to Kurento media server and from the media server all of the media will be transferred to clients.
WebRTC API deals with signaling in order to gather client's information (information about client's machine in order to check compatibility with WebRTC).

2.5.3.2 Screen sharing (Video capture of the presenter's screen)

Implemented using WebRTC API with the call mediaDevices.getOutPutMedia() function call, we can request for a selection of different captures, such as browsers tab capture, desktop capture or application window capture. After the capture is set up with the WebRTC API we can redirect the stream to clients through peer to peer or to a media server.

2.5.3.3 Switching presenter to a viewer client

This feature is a bit complicated due to the fact that the presenter has the ability to request one of his connected clients to share his screen capture and web cam. The process to implement this would be writing a script that when the broadcaster requests a client to broadcast his screen, the broadcasters screen capture is being closed. A request is sent to the client to share his screen the stream is then being captured the same way as the previous presenter and the stream is being sent to the media server, from the media server the media is transferred to all connected clients.

2.5.3.4 File upload

File upload is done with html file API and basic JavaScript libraries. Uploaded files are stored on the server.

2.5.3.5 PDF render

Using PDF.js we can render uploaded pdf files within the web application, another solution for rendering a pdf would be using an <iframe> but due to not all browsers supporting <iframes> PDF.js is safer solution in rendering pdfs.

2.5.3.6 Group chat

With the use of RTCDataChannels API which is part of WebRTC and JavaScript we can create a text based group chat for our application.

2.5.3.7 Notification of connected viewer clients

I will write a JavaScript for displaying an image based on whether the client is connected, not connected and invited. This implementation contains tracking of clients that have joined the session and displaying with appropriate image.

Education TV

2.5.3.8 Login and Registration

The functionality between login and registration will be done with the use of HTML forms and PHP to connect to MySQL database and authenticate users.

2.5.3.9 Scalable UI for mobile devices

This will be implemented using modern web CSS technologies such as bootstrap and grids in order to scale to phones appropriately but only the students view will be implemented for this, due to the reason that WebRTC Screen sharing does not work with android devices so there is no reason to implement scalable UI for broadcaster as broadcaster won't be able to run online classes from the phone it self.

2.6 Conclusion

In conclusion within this chapter I had discussed the research that had been performed in order to gain information about the requirements for the purposes of implementing this project. I gained insight on what other similar type of software had been offering which provided me with additional information when it came to finalizing my own requirements for delivering my system. The next chapter discusses how my research findings influenced the design process for this project.

3 Design

3.1 Methodology

My approach to this project is completing the design first for the web application with the focus on the implementation of the main communication system and ability to display lecture slides or screencast lecturer's desktop. Then I will prioritize features that have a high risk of implementation and require longer development time than other features of the system due to their higher risk. My approach would require lot's of testing throughout the implementation of the project therefore I considered two software methodologies in particular.

Incremental Development methodology this software methodology breaks down the project into smaller objectives in order to provide ease of change during development as each part of the project is developed separately. This development methodology applies well to my project because, my project consists of multiple functionalities that don't really depend on each other's development process they can be implemented independently, therefore it would be quite easy to identify each objective separately for the project's implementation. Incremental development consists of identifying each of the project's objectives and forming a mini waterfall model each objective. The drawback of this is that you can only be working on a single objective before moving on to the next objective for the project, this application of the software development methodology could cause a problem, as my projects individual functionalities have to be independent of each other rather than follow a particular order of implementation. Also the mini waterfall models don't allow for change of design during implementation process therefore the initial software development methodology is not suitable for the project as I want each individual functionality development to be free of other project objective completion. For incremental development initial software concept or a prototype is implemented into a final product, this approach is well suited for the development of my project as i want to develop multiple prototypes as I go in order to test each functionality of the project. This software development methodology supports testing of functionality after each implementation of the project, this applies well to my project as I want to make sure that each functionality is working well within my prototypes with multiple users, I will be doing multiple tests with different users in order to make sure each objective is completed well before moving on to the next objective within my project. But this development methodology prevents change to each individual objective of the project, because each individual objective uses the mini waterfall model and as waterfall model it does not allow to move upstream to a preceding phase, therefore after testing phase of an implemented objective i won't be able to go back to design phase or implementation to change it after testing and user feedback. There are variations of waterfall model that allow for phase upstream but they are not part of incremental development methodology therefore cannot be used.

The other methodology that I researched was Agile Development. Agile development is great for small projects that require quick delivery and don't have a set final requirements for the project. Agile development methodology allows for changes to be made late in the development process, working features of the software are delivered within weeks rather than months. Agile development uses the iterative development method what this means is that each task is broken down into small increments with minimal planning in mind. Each iteration begins with planning, requirements analysis, design, coding, unit testing and acceptance testing these iterations usually run from one week to 4 weeks. At

the end of each iteration a small piece of software is being delivered usually bug free due to testing being done at the end of each iteration.

Agile development is a perfect methodology for my project, because my solution does not have a clear final product vision due to ever changing technologies that I will be using and I need to allow for changes to be done throughout the development. If I do not allow for changes my projects features might not be functional due to ever changing WebRTC. Also with agile development extra features can be incorporated during the development process due to each iteration having a design phase. Agile development focuses on the software aspect of the project which is coding rather on design documents and planning, which I much more prefer. I will be using agile development methodology for my project because you cannot plan for everything with this project and I will be performing extra research throughout the development process of the project when needed as updates are being made to WebRTC and Chrome.

If time runs short, I have high risk and lower risk implementation objectives. I will focus on the high risk implementations at first as these high risk objectives would be the core aspects of the project and additional functionalities would be just low risk implementations that shouldn't require much time to implement but are not part of the core requirements for the project example of such would be a well built GUI. If I run out of time, but due to prioritization of core features of the system I should have the system fully working developed, graphical user interface is lowest prioritization for the project therefore it would be left out from development if time ran short.

3.2 User Interface

Low fidelity prototyping was completed in order to get the view of the graphical user interface for the system. This gives an overview of how the user interface should look like once the project is completed. Due to this being a low fidelity prototype the interface is not file and is likely to change with the final delivery as redesigns are likely.

3.2.1 Low fidelity Prototypes

Figure 3-1 Shows a low fidelity prototype that had been done in early stages of UI design, the low fidelity prototype displays a login webpage.

Education TV

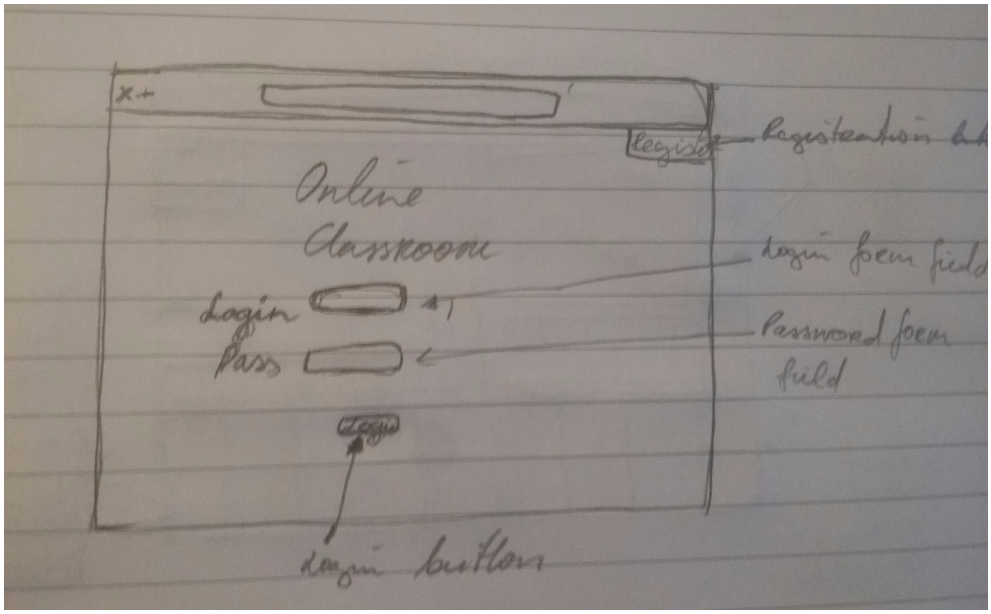


Figure 3-1 Login Page

The figure 3-2 bellow displays a low fidelity prototype of a registration page.

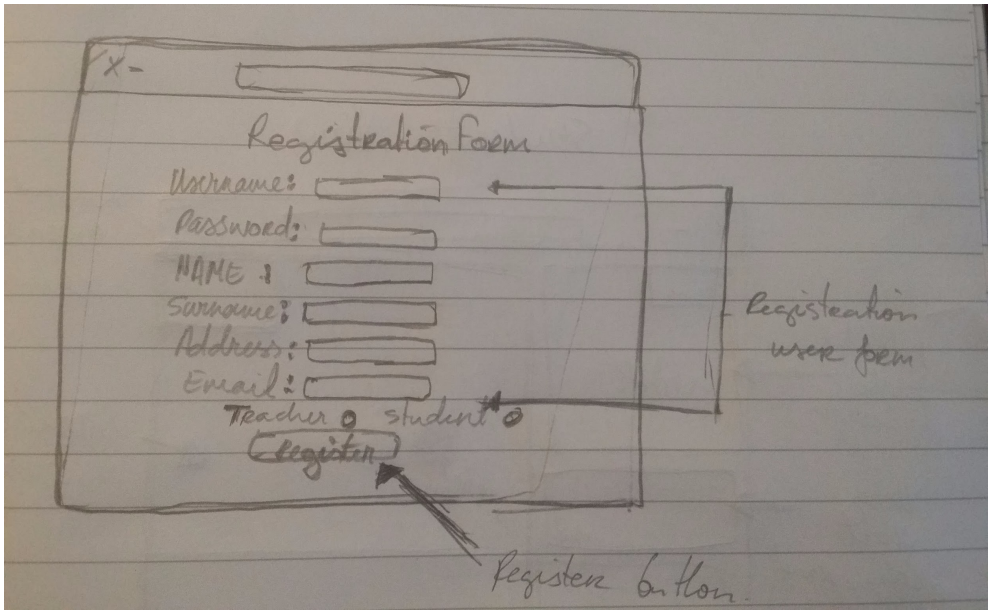


Figure 3-2 Registration page

The figure 3-3 displays the early design of the present page as a low fidelity prototype of a presenter view.

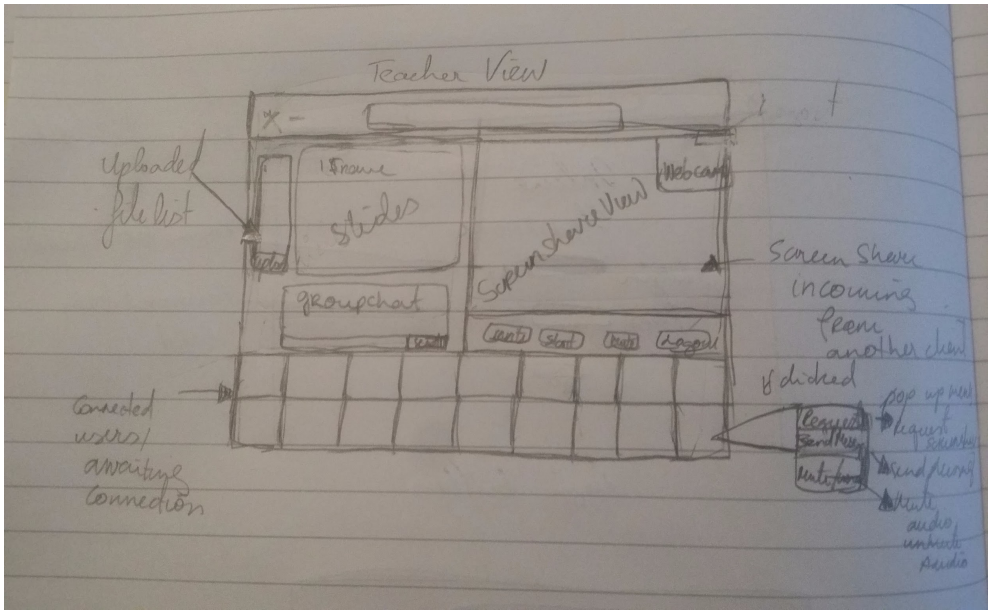


Figure 3-3 Presenter page

The figure 3-4 dispalys an early prototype of the viewers displays in a form of a low fidelity prototype.

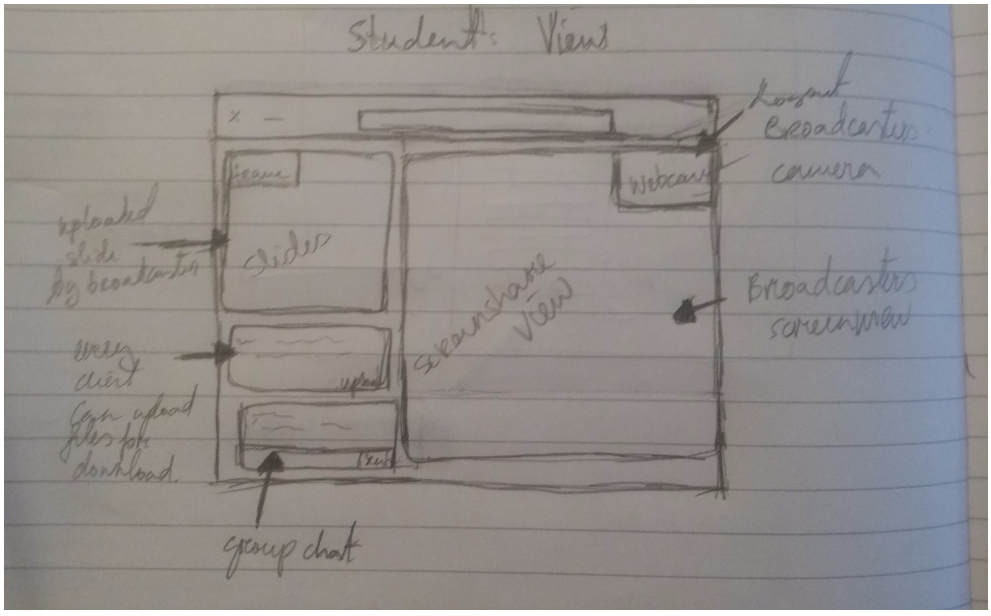


Figure 3-4 Viewer page

The figure 3-5 displays an early prototype view of the android scaling of the web application for the viewer.

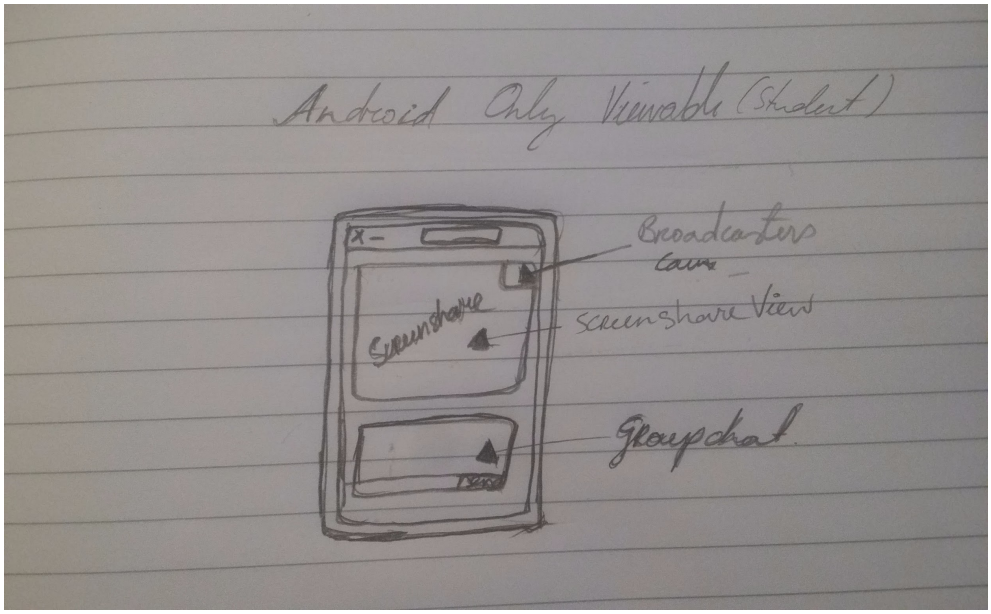


Figure 3-5Mobile phone page

3.2.2 Medium Fidelity Prototypes

Medium fidelity prototyping was completed using Fluid, this prototype was created in order to inspect how the user interface would function and get a better look at the user interface. Fluid UI allows for medium level interface design for web applications with numerous widgets available at designer's disposal.

Education TV

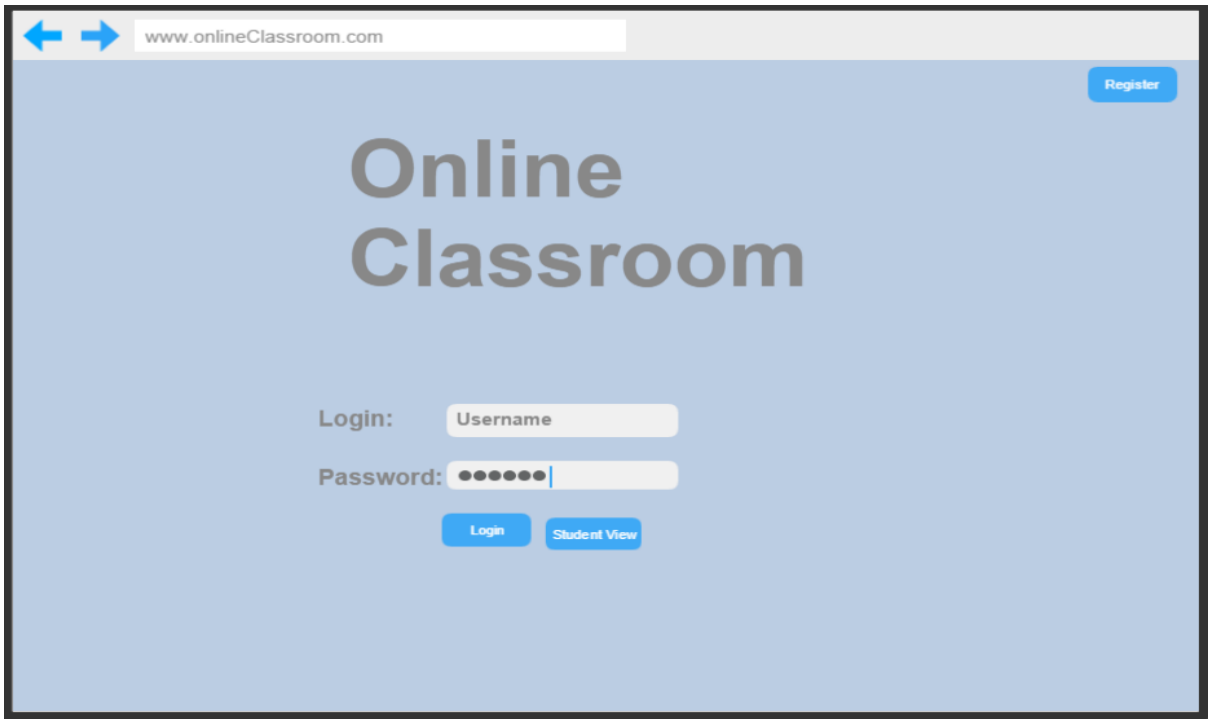


Figure 3-6 Login Screen High Level Prototype

The figure 3-7 displays a medium fidelity prototype for the presenters view created in fluid.

Education TV

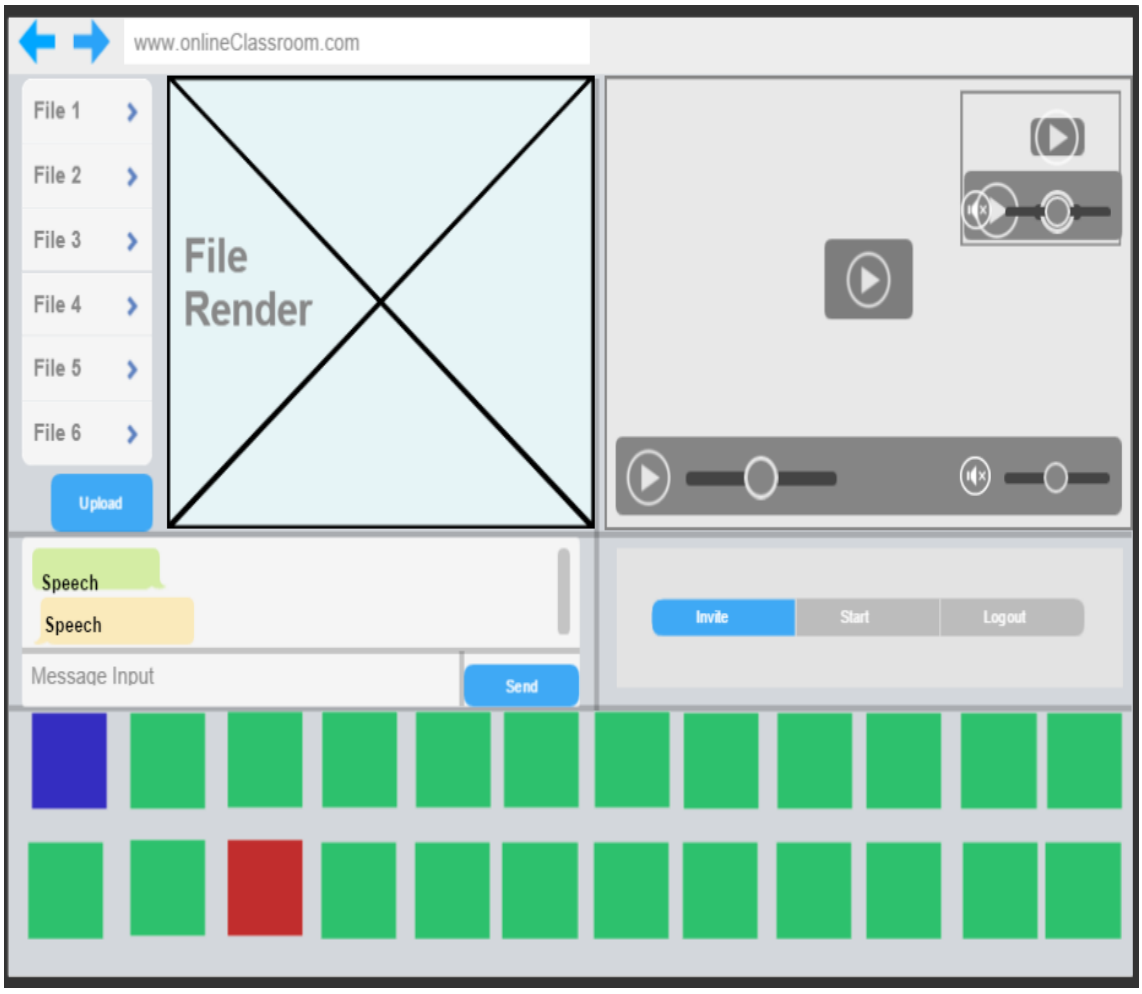


Figure 3-7 Presenter page medium level prototype

Figure 3-8 bellow shows a medium fidelity prototype for the viewer client, which primarily focuses on increasing the video size in order to deliver the most of viewing experience for the viewer. Within this webpages we can see that the video elements take the largest portion of the website.

Education TV

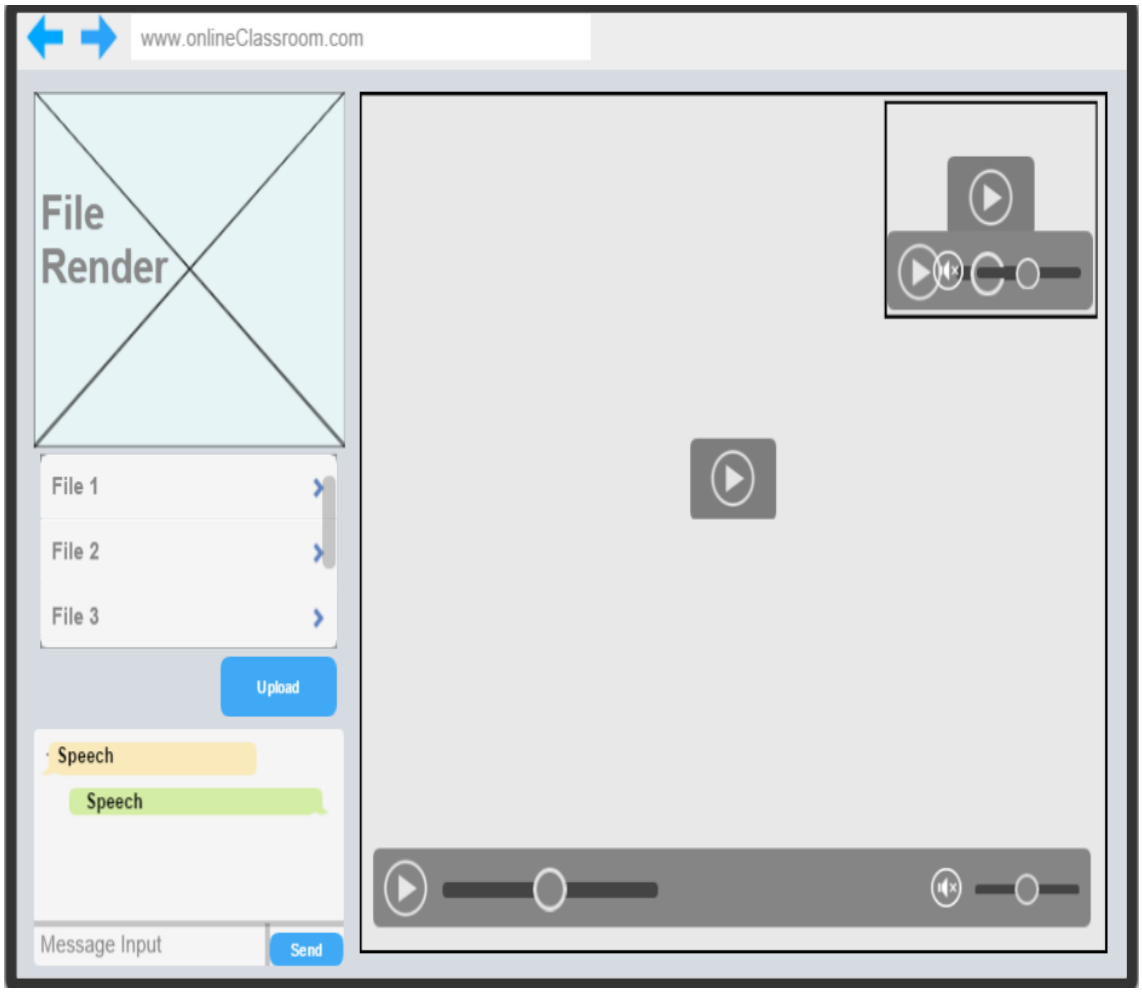


Figure 3-8 Viewer page medium level prototype

3.2.3 High Level UI

Figure 3-9 displays a high level prototype of viewer page which has a strong resemblance to medium level prototype as video elements are focused on here the most and are reflected both in medium fidelity prototype as well as high level prototype.

Education TV

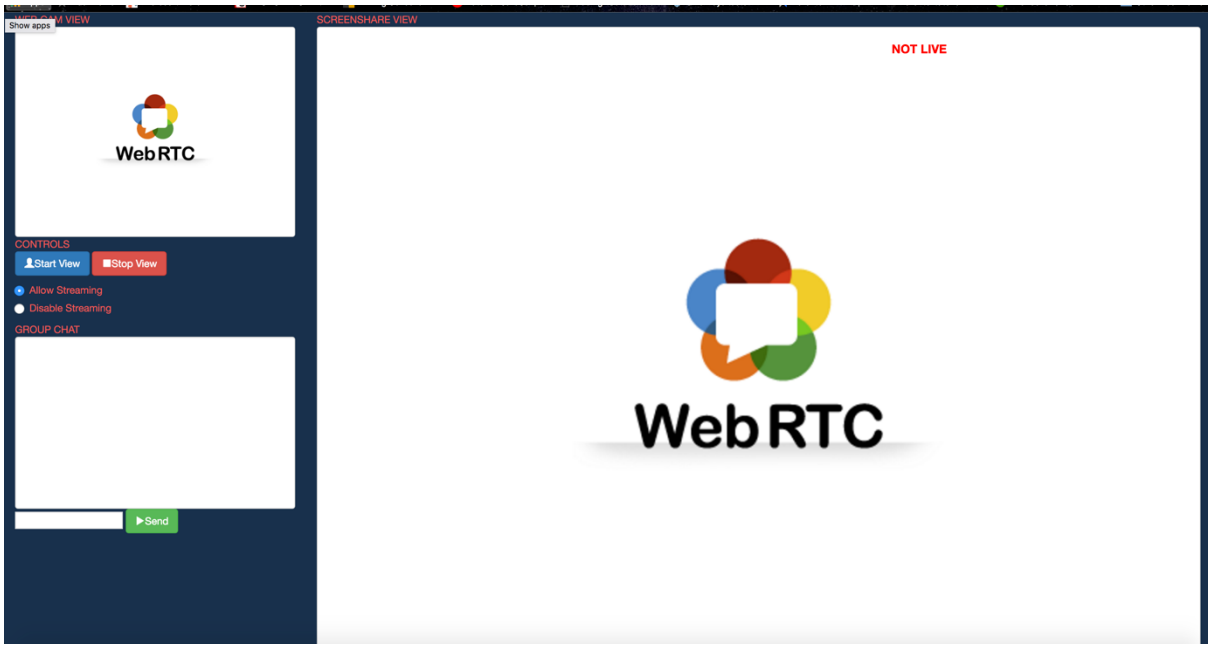


Figure 3-9 Viewer Page

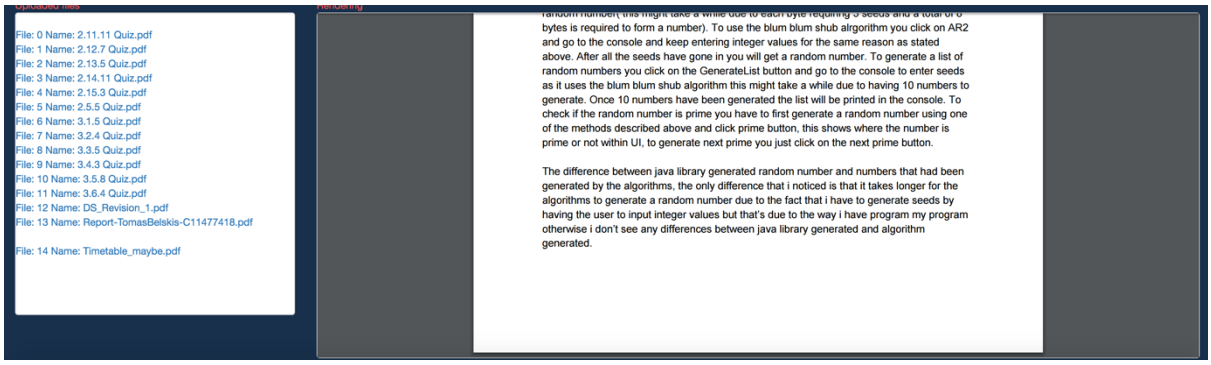


Figure 3-10 Viewer Page continued

The figure 3-11 displays high level prototype for the presenter page, which displays file rendering window as well as presenters capture playback.

Education TV

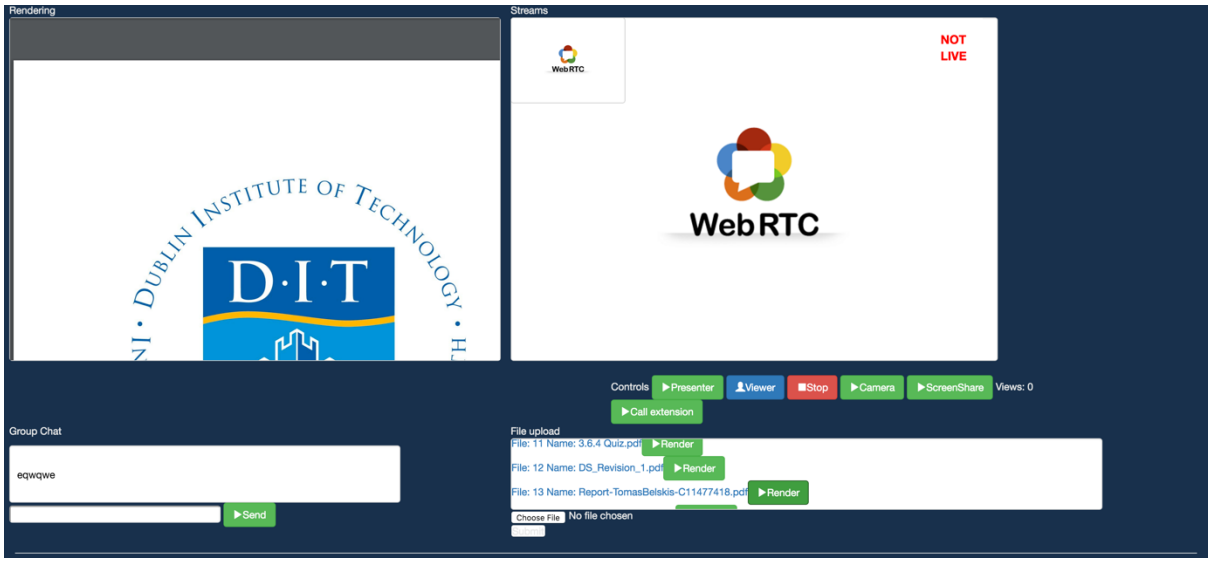
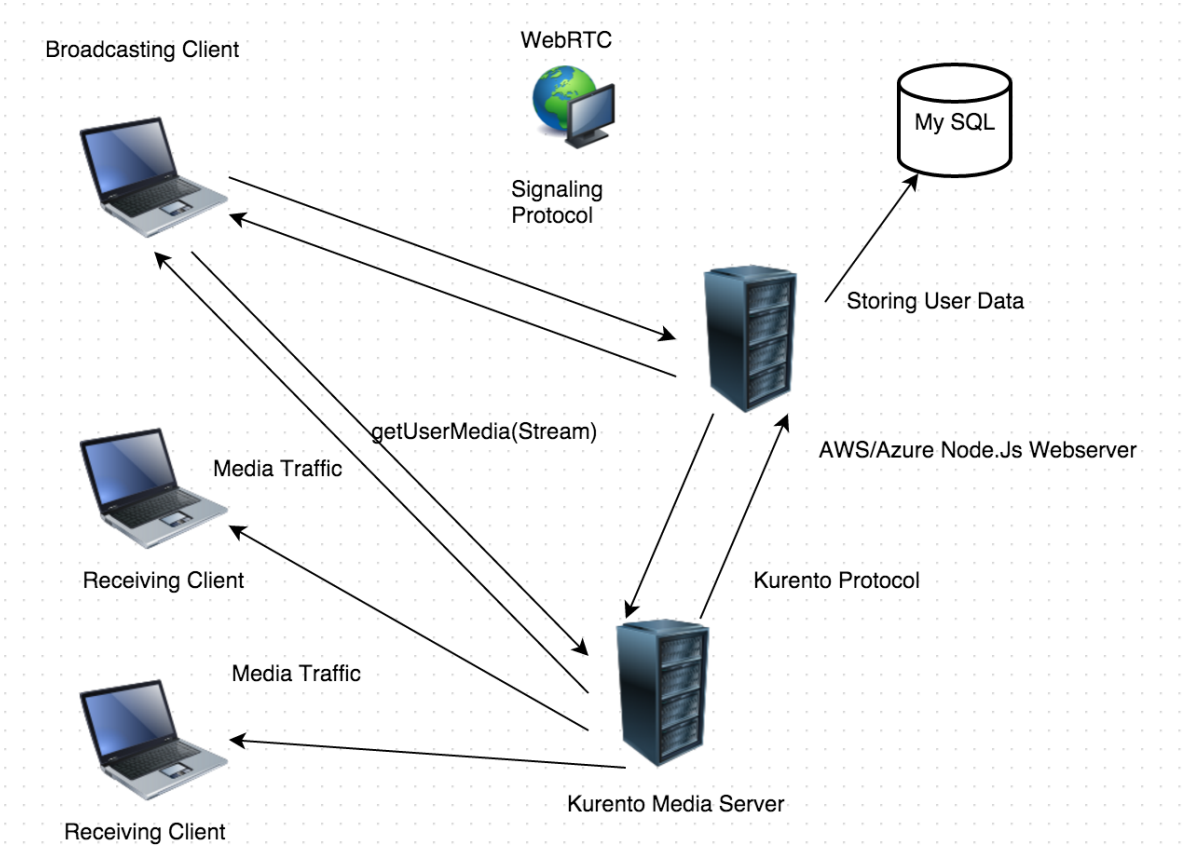


Figure 3-11 Presenter

3.3 System Components



Technical architecture diagram above displays all of the components within the system, with a sample getUserMedia call. The broadcasting client signals the server using WebRTC signaling protocol this returns information on the Clients machines that are connecting to the session. The Kurento media server interacts with Node.Js server that has been hosted on Amazon AWS through Kurento protocol this is to receive the client information. The broadcasting client then directly streams its video capture

media to Kurento server. Kurento server delivers the media traffic to receiving client viewers. Node.js server holds a database for storing user data for the purpose of authentication of users.

3.4 Source Code Layout

The application makes use of 2 servers for the basic model and one the server code implementation is cloned to create a different different channel.

The main broadcasting application server code, this server application deals with broadcasting.

- Web Server 1
  - /node_modules
    - This directory contains all of the node modules required for this webserver
  - Server.js
    - This is the server code that handles all of incoming messages from clients and set ups media pipeline with webrtc endpoints for media streaming.
  - /static
    - /bower_components
      - Javascript libraries are stored here that are used here within the application
    - Bower.json
      - Contains information about the list of libraries that had been installed
    - /css
      - contains css files for different pages
      - index.css
      - kurento.css
      - observer.css
      - recordings.css
      - viewer.css
    - /img
      - Contains images files
    - /js
      - Contains client side control javascript
      - /index.js
      - /recording.js
      - /control.js
      - /Screen-Capturing.js partially edited and referenced later on at section 4.2.2.2
      - /viewer.js
    - /pdfs
      - Uploaded pdfs stored here
    - /recordings
      - Recordings are stored here
    - /uploads
      - All other uploads are stored here
    - Index.html
    - Observer.html

        - Recordings.html
        - Test.html
        - Viewer.html
- Web Server 2
    - /Node_Modules
        - Contains modules required for server functionality
    - server.js
        - contains server code required for application
    - /static
        - /bower_components (contains JavaScript libraries used with the application)
            - /bootstrap
            - /jQuery
        - /images
            - contains images used
        - /js
            - channels.js
        - /views
            - channels.html
        - index.html
        - /css
            - channel.css
            - index.css

3.5 Features and Use Cases

General user use case:

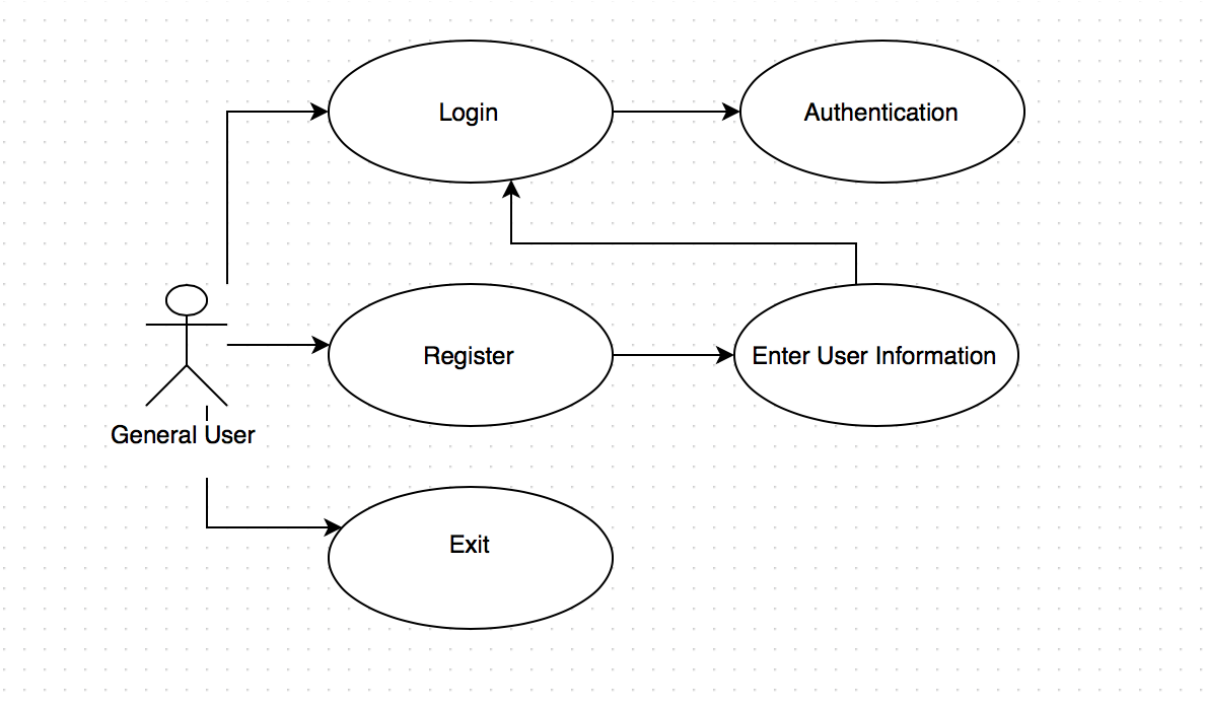


Figure 3-12 New User Use Case

Education TV

The general user is defined by entering the main domain of the web application, this allows the user to Login, Register or just exit the current webpage. If it's a new user then the user is able to register by clicking the register button and he will be forwarded to a registration form, once the registration is complete this user will be able to login using his username and password, this will authenticate the user and forward it to logged in user page.

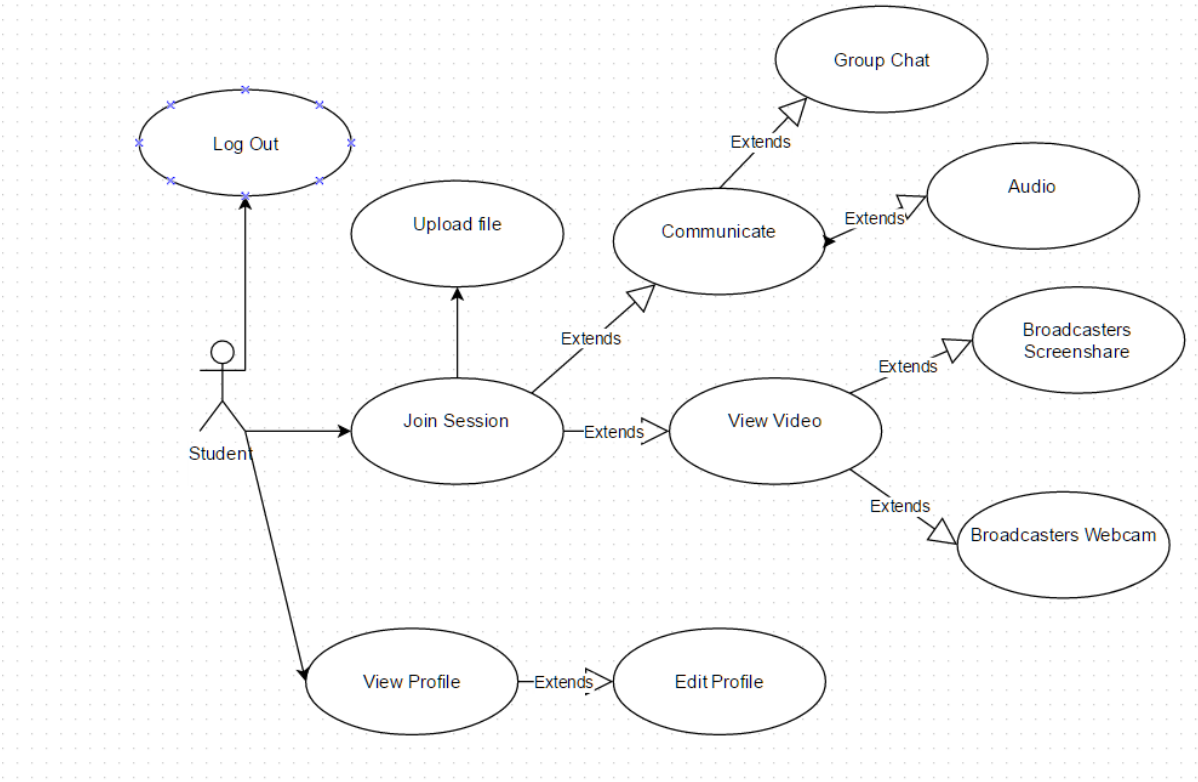


Figure 3-13 Viewer (Student) Use Case

The student user has multiple functionalities. The basic functionalities are logging out of the service, viewing his user profile and editing profile, joining a lecture session upon invitation of the lecturer allows for the student to upload files, communicate with everyone in the current session with the use of group chat, or on request by the broadcaster unmute himself in order to voice communicate. The student that has joined a session is able to view video that is being broadcasted by the lecturer (web cam and screen share).

Education TV

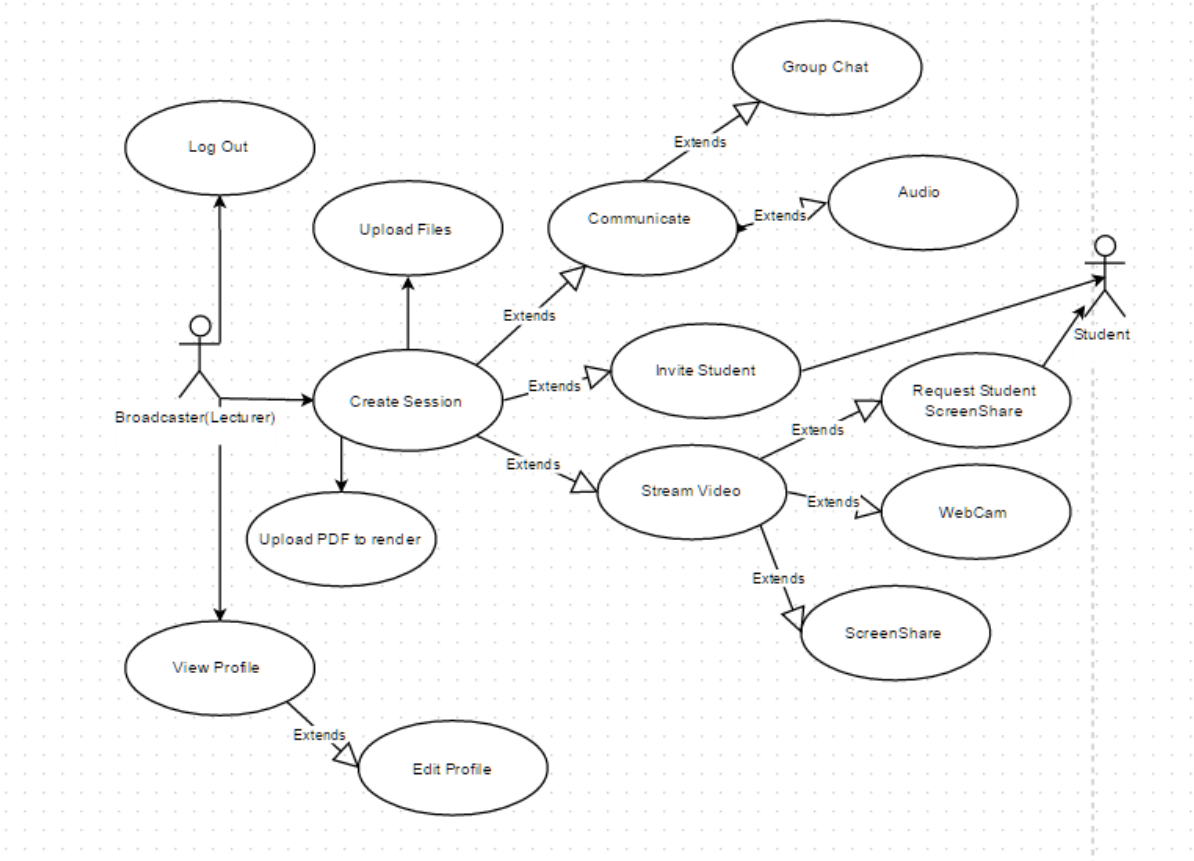


Figure 3-14 Presenter (Lecturer) Use case

The lecturer has similar base abilities as the student. The Lecturer can logout from the system this would stop the current session that is live, he is able to view his profile and edit it accordingly. The lecturer creates a session upon logging in once the session is created, the lecturer can invite students to join his session, upload files, display lecture slides (in pdf format). Upon creation of a session the lecturer automatically broadcasts webcam feed to anyone that has joined the session, and is able to start screen sharing video stream. The lecturer also can request a student to screen share the video of his screen. The lecturer can communicate with the use of audio and group chat.

Education TV

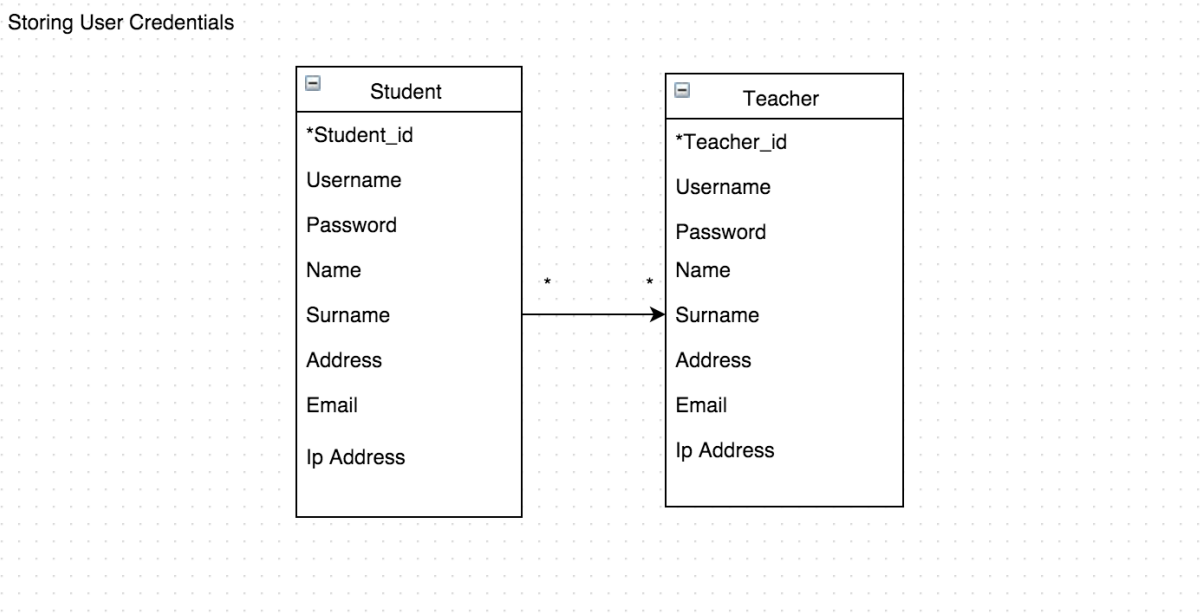


Figure 3-15 Database for user credentials

The diagram that shows the two user group tables for a database in order to store user credentials. Will need to undergo another set of design phase due to not know which fields exactly will be required for implementing features such as group chat.

3.6 Conclusion

In this chapter it had been discussed the design process that had been taken based on the research and analysis that had been performed in chapter 2. Prototypes had been produced from low level to high level prototypes these show the general views for the web application. Other aspects of design had been discussed here such as use cases and database erd. Even though database hadn't been implemented, it had been design in mind for authentication which hadn't made to final implementation. In the next chapter system components are discussed in detail.

4 Architecture and Development

4.1 Overview

Overview of the system, the systems consists of two webservers and a MCU (Media control unit). The main server broadcasting server is hosted on Amazon AWS EC2 Ubuntu server using node.js web server and Kurento media server used as MCU for the project. The second server which is the channel server is hosted on Azure Ubuntu server using node.js as the webserver. The recordings and uploaded files are stored locally on the first broadcasting server. The communication done between the client and server is done through a web socket and also the communication done between channel server and broadcasting server is done through a web socket that is created on the channel server. The broadcasting server is secured using fake certificates in order to impose HTTPS server protocol, this is requirement for use of WebRTC in order to secure video streams and notify client that it's a secure connection. The channel server is a typical HTTP node.js server. Figure 4.1 displays the components of the system.

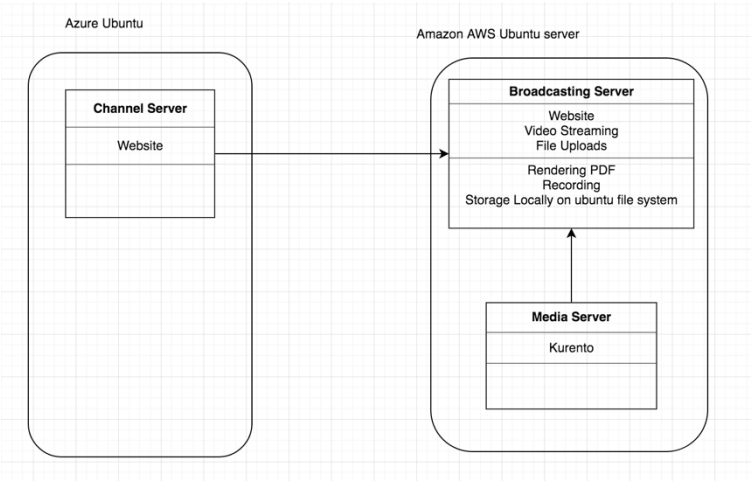


Figure 4-1 System Components

The system components can be seen in the Figure 4.1, this shows a single channel setup overall system can accommodate multiple channels if the broadcasting server is replicated and set up appropriately. The channel server serves the purpose of having navigation of each channel and it also serves as providing information for viewer clients when the channel is online or offline and provides the viewer client with navigation to channels recordings. What the system would look like with with multiple channel setup is showed in figure 4.1.2.

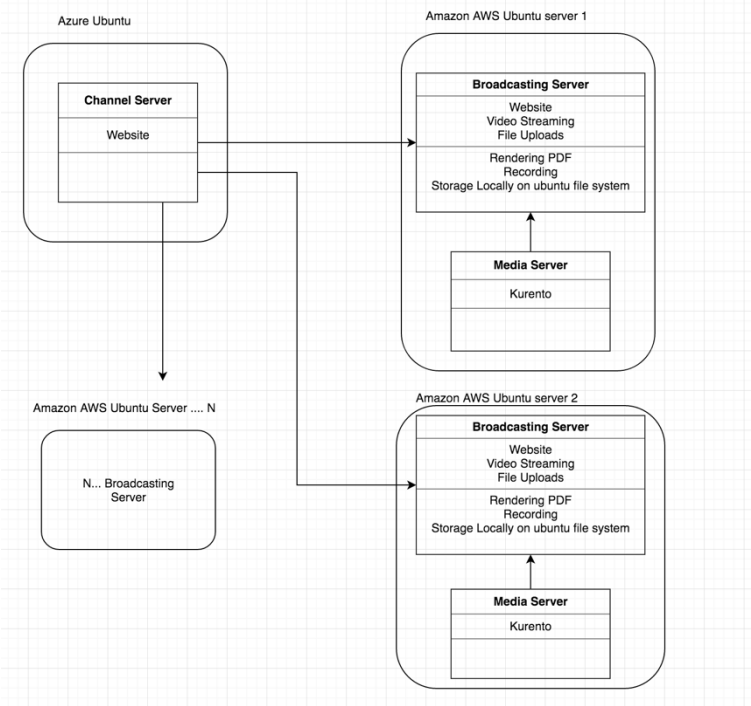


Figure 4-2 Multiple channel support

The figure above displays multiple channel support for the project where it can support of up to N channels by replicating the server setup of broadcasting server over to another Amazon AWS Ubuntu server.

The architecture consists of a Three tier architecture, client server web model. The development model for most part use MVC (Model view control) programming model, some case it doesn't quite follow it, this is reviewed in 4.2. Components including detailed information of each decision process and development of each component within the system.

4.2 Components

4.2.1 Web Servers

4.2.1.1 Broadcasting Server

The broadcasting server has been hosted on Amazon AWS EC2 Ubuntu 14.04 LTS. During the planning phase I had decided that I would be using Amazon AWS to host my web application, the reason behind using the Ubuntu distribution was due limitations of Kurento (MCU) only being available for working on Ubuntu 14.04 LTS. Node.js was chosen to be the Web server that as it worked well with WebRTC and the examples that I tried during research phase had used Node.js locally which allowed me to gain early experience using Node.js leading me to further use it as a light weight web server throughout the project.

The Kurento Media server requires a large number of ports to open therefore the broadcasting server cannot be running on Azure as azure has a limit of ports that can be open due to not having the functionality of setting port ranges.

Security Group: sg-1c256778

Description | Inbound | Outbound | Tags

Edit

Type	Protocol	Port Range	Source
Custom UDP Rule	UDP	4080	0.0.0.0/0
Custom UDP Rule	UDP	3478	0.0.0.0/0
HTTP	TCP	80	0.0.0.0/0
Custom UDP Rule	UDP	49152 - 65535	0.0.0.0/0
Custom TCP Rule	TCP	8888	0.0.0.0/0
SSH	TCP	22	0.0.0.0/0
Custom TCP Rule	TCP	4080	0.0.0.0/0
Custom TCP Rule	TCP	1000	0.0.0.0/0
Custom TCP Rule	TCP	3478	0.0.0.0/0
Custom TCP Rule	TCP	8443	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0

Figure 4-3 Amazon AWS EC2 Instance Open Ports

In figure 4-3 we can see that the port range 49152-65535 is set this port range is used by Kurento media server for UDP transfers. The port 8443 is used by the application, port 8888 is the port that Kurento media server is running on.

Broadcasting server uses multiple libraries in order to simplify implementation. The communication between client and server is done with the use of WebSocket hosting on location of the webserver and a particular port.

```
83  //creating https server on port
84  var server = https.createServer(options, app).listen(port, function() {
85      console.log('Application Server has started! ');
86      console.log('Open ' + url.format(asUrl) + ' with a WebRTC capable browser');
87  });
88
89  // new websocket server
90  var wss = new ws.Server({
91      server : server,
92      path : '/broadcasting'
93  });
```

Code Extract 4-1 - Broadcasting server.js websocket creation

The code snippet 4-1 shows the creation of https server running on a custom port, the port was set to 8443 as one of the available ports to use. Following the creation of web server, a websocket server was required to be created this websocket server runs on the node server as you are passing it as an argument within JSON data type and the path is used to make it more distinguishable for the purpose of of the websocket is set to /broadcasting.

```
1
2   var ws = new WebSocket('wss://' + location.host + '/broadcasting');
```

Code Extract 4-2 Broadcasting server index.js websocket

The code snippet 4-2 shows the creation of the websocket on client side, referencing the location.host which is the host location of the node.js server that is running on server side.

Through this websocket all the communication between the client side and server is done, the messages that are being sent through the socket are in form of a string, before messages are sent they are formed

as JSON, the JSON messages are then stringifyied before being sent through the websocket and parsed on receiving side back into JSON for easy of managing the information that is received. The sever handles client messages through the use of switch case statement.

```
    //message processing
    ws.on('message', function(_message) {
        var message = JSON.parse(_message);
        console.log('Connection ' + sessionId + ' received message ', message);
        //dealing with messages passed on websocket
        switch (message.id) {
        case 'presenter':
            startPresenter(sessionId, ws, message.pSdpOfferCam, message.pSdpOfferScreen, message.control, function(error, pSdpAnswerCam, pSdpAnswerScreen) { // starts presenter by sending sessionID, websocket and sdpoffer
                if (error) {
                    return ws.send(JSON.stringify({
                        id : 'presenterResponse',
                        response : 'rejected',
                        message : error
                    }));
                }
                ws.send(JSON.stringify({
                    id : 'presenterResponse',
                    response : 'accepted',
                    pSdpAnswerCam : pSdpAnswerCam,
                    pSdpAnswerScreen : pSdpAnswerScreen
                }));
            });
            break;

        case 'viewer':
            startViewer(sessionId, ws, message.vSdpOfferCam, message.vSdpOfferScreen, function(error, vSdpAnswerCam, vSdpAnswerScreen) {
                if (error) {
                    return ws.send(JSON.stringify({
                        id : 'viewerResponse',
                        response : 'rejected',
                        message : error
                    }));
                }
                    ws.send(JSON.stringify({
                        id : 'viewerResponse',
                        response : 'accepted',
                        vSdpAnswerCam : vSdpAnswerCam,
                        vSdpAnswerScreen : vSdpAnswerScreen
                    }));
                });
            break;

        case 'observer':
            startObserver(sessionId, ws, clientSessionIds);
            break;

        case 'chat':
            var chatMsg = message.chatMessage;
            wss.broadcast(JSON.stringify({
                id : 'chatResponse',
                chatMessage : chatMsg
                }));
            break;

        case 'streamTypeRequest':
            clientsList[sessionId].send(JSON.stringify({
                id: 'typeResponse',
                streamType : videoStreamType
```

Code Extract 4-3 Broadcasting Server server.js Message Handling

The code snippet 4-3 shows a small portion of the messages that server handles between client and responses or functions that are being invoked due to certain messages received from the client.

The Kurento media had been installed on Ubuntu web server and running on port 8888, the broadcasting server retrieves the Kurento client, that is later used for redirecting media traffic to Kurento media server when the presenter is being started in order to create a media pipeline that the media stream will be sent through into Kurento Media Server.

```
// Recover kurentoClient for the first time.
function getKurentoClient(callback) {
    if (kurentoClient !== null) {
        return callback(null, kurentoClient);
    }

    kurento(argv.ws_uri, function(error, _kurentoClient) { //function that is part of instanciating a kurento client
        if (error) {
            console.log("Could not find media server at address " + argv.ws_uri);
            return callback("Could not find media server at address" + argv.ws_uri
                    + ". Exiting with error " + error);
        }

        kurentoClient = _kurentoClient;
        callback(null, kurentoClient);
    });
}
```

Code Extract 4-4 Broadcasting Server server.js Kurento client

The code snippet 4-4 shows the Kurento client function that invokes the Kurento Media server that is running on Ubuntu the figure 4-5 shows the location of where the Kurento media server is running. This function returns the instance of Kurento client which is later used for media streams. The usage of Kurento client will be further looked at in section 4.2.2 Media Streaming.

```
//sets kurento client url 8888, kurento client running on port 8888, and access url both on port 8443
var argv = minimist(process.argv.slice(2), {
    default: {
        as_uri: 'https://52.49.81.164:8443/',
        ws_uri: 'ws://52.49.81.164:8888/kurento'
    }
});
```

Code Extract 4-5 Broadcasting server server.js Locations

The code snippet 4-5 shows the locations of where the different webservers are running. The node.js https server is running on port 8443, and the websocket that we get the Kurento client instance from is running on port 8888 path /Kurento, this websocket URI used in getting the Kurento client using the function in code snippet 4-4.

The MVC architecture was used minimally within the application, to achieve MVC architectural patter ExpressJS API was used which is a MVC type framework for web applications developed using node.js.

The server code was built with the use of Kurento documentation, and some of the code had been used and modified from the Kurento one2many example.

4.2.1.2 Channel Server

The channel server is set up on Azure cloud services. It uses similar set up to the broadcasting server as it is also running Ubuntu 14.04 LTS server and Node.js server is used with websocket for communication. The purpose of the channel server is to keep track of channel information when the channel goes live and offline also to forward users to right page it servers as a front end control for the application. Start page is hosted here as well as channel selection page. The channel server holds a websocket that the client uses to communicate and also this websocket is used on the broadcasting server where the broadcasting server sends a message to channel server that the presenter has went live which changes the channel to a live channel and offline when the presenter stops presenting.

```
var server = http.createServer(app).listen(port, function () { console.log('Listening on ' + server.address() +':'+ server.address().port) });

var wss = new ws.Server({
    server: server
    });
```

Code Extract 4-6 Channel Server server.js Server creation and websocket creation

A http server is created in code snippet 4-6 and a websocket server. This websocket is then recreated on the broadcasting server for communication.

```
var channelsURL = 'project-vm.cloudapp.net:4080';

var chWs =  new ws('ws://' + channelsURL);
```

Code Snippet 4-7 Broadcasting Server server.js communication with channel server

The code snippet 4-7 shows the websocket that is being created to complete the websocket communication between the two servers. This is used to get information when the presenter is live. On load of the channels webpage that is hosted on the channels server as message is sent to broadcasting server to retrieve information whether a presenter is presenting at that moment or not. The broadcasting server has a value that holds the information if the channel is live or not this value changes every time presenter is started or stopped correspondently. The channel server asks for this value and broadcasting server returns it, the channel server changes the image on the client side page channels to either an image of live channel or an offline channel. Initially the broadcasting server sends a message to the

channel server when ever presenter goes live or offline in order to change the channel notification even if the page had been loaded.

```
ws.onopen = function(){
                    console.log('connection established');
                        getChannelState();
                    };
```

Code Extract 4-8 Channel Server channel.js

The code snippet 4-8 shows the function that gets invoked when the connection has been opened on the websocket, this connection is opened when the channels html page is loaded. The ws is a reference to the websocket used for communication with channels server. The function getChannelState () is invoked when the websocket is opened for communication.

```
function getChannelState(){
    var message = {
        id : 'channelStateRequest'
    };

    sendMessage(message);
}
```

Code Extract 4-9 Channel.js Message to server

The code snippet 4-9 shows a message that will be sent with the id 'channelStateRequest' to the server, the sendMessage function just stringifies the JSON object and invokes the ws.send() which sends the message onto the server.
The server side then handles the message received from the client. This can be see in the figure 4-10.

```
143             case 'channelStateRequest':
144                clientList[broadcastingServerSessionID].send(JSON.stringify({
145                    id : 'getChannelState'
146                }));
147                break;
148
```

Code Extract 4-10 Channel server.js

The code snippet 4-10 upon receiving message with id channelStateRequest sends a message to the broadcasting server with the id getChannelState which then when message is received on the broadcasting server it returns a message to the channel server with either id live or offline. Depending on the message returned server sends a message to the client in order to modify the channel image to offline or online.

```
var parsedMessage = JSON.parse(message.data);
console.log('Received message from channel server: ' + message.data);

switch(message.id){
    case 'getChannelState':
        channelState(presenterState);
        break;
```

Code Extract 4-11 Browser server.js handle message from channel server

The code snippet 4-11 displays handling of message that is received from channel server for request of channel state, presenterState is data variable containing live or offline value depending if the presenter is currently presenting or not. The channelState function is invoked which sends state of the presenter back to channel server.

Once the server receives the message it passes on the state to a client and the javascript on the client changes the image for the channel depending on the state variable this can be seen in figure 4-12.

```
49      switch(parsedMessage.id){
50
51          case 'live':
52          console.info('Channel is going live! ');
53              changeChannelState(parsedMessage.id);
54              break;
55
56          case 'offline':
57          console.info('Offline is going live! ');
58              changeChannelState(parsedMessage.id);
59              break;
60
61          default:
62              console.error('Unrecognized message: ' + message.data);
63              break;
64      }
65  };
```

Code Extract 4-12 Channel.js handling server message

In the code snippet 4-12 the state of channel is being changed depending on the message that is received from the server, the function changeChannelState changes the image on channels.html to offline or online

The channels server wasn't in the original plan for the web application, but due to the design changing from the application only being as education application, to a multi channel application with each channel being hosted on it's own Amazon AWS server, the requirement for a channel server that has references to each channel was required.

4.2.2 Media Streaming

Media streaming is the core component of this project, there are two forms of media being streamed video of client's webcam and screen share video of the client's screen. The presenter client is the client that is able to select the option of stream that he wants to broadcast to connecting viewer clients. The video streaming uses HTML 5 <video> tags for video playback. WebRTC API is used in order of

getting the users media through the browser. Once the source of the media is received then a peer connection is being negotiated and a media pipeline is being created for the media to be transferred over the Kurento media server. The viewer client's that join in order to view the stream, send a signaling request to create a peer connection between the presenter client once successful connection has been negotiated with the right Session Description Protocol Answers being received by the clients. The adapter.js API is used to merge all the different browser implementation for WebRTC negotiations in order of support for multiple browser implementation. Kurento-utils.js API further simplifies the implementation of WebRTC and reduces the amount of code you have to write, kurento-utils.js is a wraps the WebRTC code and simplifies its implementation.

The basic flow of WebRTC in order of capturing media and setting up a peer connection. In order to get access of client's web camera a basic getusermedia call is being invoked by default this requests access to client's webcam through the browser the user can deny the request or allow it through, this specifies if the media is to be received video only or audio only or both video and audio. This sets the local description of the media that is being captured. Next a peer connection needs to be created which is created by sending by invoking RTCPeerConnection this sends iceCandidate with the media stream information and clients fingerprint to the signaling server in the project Node.js is used as signaling server these iceCandates are then queued at Node.js server. Session Description Protocol offer is sent to the server as well which contains media type being to be transmitted over UDP this is a requirement for Kurento media server. On the server side the sdpOffer is being processed by the Kurento client and Kurento client generates the sdpAnswer which is sent back to the client with the Kurentos iceCandidate after the clients iceCandidate had been registered with Kurento client. Client receives the iceCanditates and sdpAnswer, they are then processed on the client side and if they are valid a peer connection has been established and a media pipeline created with a webRtcEndpoint. The process is then repeated for when the viewer client requests to view the stream except of the viewer being requested its media it sets the location of where the media to be sent to which is the html video tag within the html page. Server side creates a viewer endpoint on the already existing presenter media pipeline into this endpoint media is being received from Kurento media server.

4.2.2.1 Web cam streaming

```
function startBroadcast(type){

    var options1 = {
                localVideo: video,
                onicecandidate : onIceCandidateCam,
            };

    if(type=='cam'){

        //alert('cam');

        webRtcPeerCam = kurentoUtils.WebRtcPeer.WebRtcPeerSendonly(options1, function(error) {
            if(error) return onError(error);
            this.generateOffer(onOfferPresenterCam);
        });
```

Code Extract 4-13 Broadcasting server index.js Webcam stream call

The code Snippet 4-13 Displays the creation of webcam stream peer. The function start Broadcast gets called when a URL is clicked which is styled as a button and depending of which button is clicked a type is passed as a parameter to the function if the button for cam stream is clicked then the type of stream is set to cam and on this type of stream a peer connection is created using kurento-utils as this is a send only peer connection it won't be receiving any media from other clients. The options that are being invoked are for local Video which is where the cam capture will be play backed for the presenter

it's a html 5 video element on the webpage, and an iceCandiate that get's created when creating the client peer connection and send to the server, an anonymous call back function is as a callback, which generates an offer and a function is passed as parameter that get's called when the offer had been generated, the onOfferPresenterCam function sends the sdpOffer that can been generated to the server.

The server then attempts to create a media pipeline, Kurento client parses the sdpOffer generates the the sdpAnswer, stores iceCandidates and sends it back to the client finishing off the connection between Kurento client and establishing media pipeline with and endpoint for media to be sent to Kurento Media server.

The viewer then creates a peer connection in receive mode specifying remoteVideo location which's html 5 video element. On the server side the offer to receive media is being parsed and an answer is generated in the process endpoint on presenter pipeline for the viewer is created and the presenter endpoint is connected to viewer endpoint letting the media to be streamed from presenting client into Kurento media server and from media server into client.

```
887         presenter.pipeline.create('WebRtcEndpoint', function(error, webRtcEndpointCam) {
888             if (error) {
889                 stop(sessionId);
890                 return callback(error);
891             }
892
893             viewer.webRtcEndpointCam=webRtcEndpointCam;
894
895             if (presenter === null) {
896                 stop(sessionId);
897                 return callback(noPresenterMessage);
898             }
899
900             if (candidatesQueueCam[sessionId]) {
901                 while(candidatesQueueCam[sessionId].length) {
902                     var candidate = candidatesQueueCam[sessionId].shift();
903                     webRtcEndpointCam.addIceCandidate(candidate);
904                 }
905             }
906
907             webRtcEndpointCam.on('OnIceCandidate', function(event) {
908                 var candidate = kurento.register.complexTypes.IceCandidate(event.candidate);
909                 ws.send(JSON.stringify({
910                     id : 'iceCandidateCam',
911                     candidate : candidate
912                 }));
913             });
914
915             webRtcEndpointCam.processOffer(vSdpOfferCam, function(error, vSdpAnswerCam) {
916                 if (error) {
917                     stop(sessionId);
918                     return callback(error);
919                 }
920                 if (presenter === null) {
921                     stop(sessionId);
922                     return callback(noPresenterMessage);
923                 }
924
925                 presenter.webRtcEndpointCam.connect(webRtcEndpointCam, function(error) {
926                     if (error) {
927                         stop(sessionId);
928                         return callback(error);
929                     }
930                     if (presenter === null) {
931                         stop(sessionId);
932                         return callback(noPresenterMessage);
933                     }
934
935                     viewerSdpAnswerCam=vSdpAnswerCam;
936
937                     webRtcEndpointCam.connect(recorderEndpoint,function (error){
938                         if(error) console.log('Viewer recording error');
939                     });
940
```

Code Extract 4-14 Broad casting server.js presenter and viewer endpoint connection

The code snippet 4-14 shows the creation of an endpoint on already existing pipeline that has an endpoint already which is used by the presenter client, the endpoint that is being created on line 887 is the viewer endpoint that will be connected to presenter endpoint further down. Line 925 shows the connection of presenter camera endpoint to the newly created camera endpoint for the viewer which completes the media pipeline for media transfer.

One of the kurento examples was used as a basis code used for streaming web cam media was from a kurento example one2many. This code was highly modified and only certain parts were used.[46]

4.2.2.2 Screen sharing (Video capture of client's screen)

For most part the screen capture streaming is the same to video cam streaming, the difference is that it requires an extension to be called once the extension is called a window selector is displayed to the client once the client selects which window he wants to screen share that window unique id is returned which is then used to form constrains for media to be streamed. Initially I attempted to develop my own screen capturing chrome extension but as it was resulting in a lot of time being consumed to get it to work I searched for an open sourced chrome extension for screen capturing, the first chrome extension that I tried did not work with the project due to unknown errors and a result of not having been able to get this extension to work and wasting a considerable amount of time in attempt to get the extension to work I looked for an alternative open source extension when I came across an extension that was developed by which is open sourced and available to use and it comes with a open source screen-capturing.js java script that calls the screen capturing extension,[47] the screen-capturing extension had been modified to suit my application requirements as the screen capturing JavaScript initially allowed for a single call to the extension and it wouldn't be recalled afterwards, it attempted to reuse the same screen id that was previously selected which is not appropriate to the usage of my application.[48]

```
}else if(type=='screen'){

    //alert('screen');

getSourceId(function(sourceId){

    var screen_constraints = {
        audio:false,
        video:{
            mandatory:{
                chromeMediaSource: 'desktop',// error ? 'screen' : 'desktop',
                maxWidth: 1920,
                maxHeight: 1080,
                chromeMediaSourceId: sourceId
            },
            optional:[]
        }
    }

    console.log("xxxx-sourceID:"+ JSON.stringify(sourceId, null, 4));
    console.log("xxxx-constraints:"+ JSON.stringify(screen_constraints, null, 4));


var options2 = {
              localVideo : screen,
              onicecandidate : onIceCandidateScreen,
              mediaConstraints : screen_constraints,
              sendSource : 'screen'
            };

            webRtcPeerScreen = kurentoUtils.WebRtcPeer.WebRtcPeerSendonly(options2, function(error) {
                if(error) return onError(error);
                this.generateOffer(onOfferPresenterScreen);
            });
    });
```

Code Extract 4-15 index.js Screen Capturing

The code snippet 4-15 shows how the screen capture peer connection is being send to created. The main different is the options that are being sent, the function getSourceId is part of the screen-capturing.js this function calls the extension and returns sourceid through a callback. When dealing with screen share media valid screen_constrains have to be set, as of now chrome does not support audio with screen capture therefore the requirement is to set audio to false, video constrains are need to be set, as this what determines the type of media that is to be streamed, media source is set to desktop and media source is also mandatory as it determines which screen is being captured. The constraints

are then set in options which are passed into creation of peer connection using kurento utils. When kurento utils comes across the send source as being screen it calls a function getScreenConstrains which had to be written by me due to kurento-utils not providing this function. By default, kurento utils only deals with web cam streams. The function was written within screen-capturing.js code snippet 4-16 shows the function.

```
function getScreenConstraints(sourceId, callback) {
    console.log("=== window.getScreenShare === " + JSON.stringify(sourceId, null, 4));
    console.log("=== window.idSource === " + JSON.stringify(window.idSource, null, 4));
    var screen_constraints = {
        audio: false,
        video: {
            mandatory: {
                chromeMediaSource: 'screen',
                maxWidth: window.screen.width > 1920 ? window.screen.width : 1920,
                maxHeight: window.screen.height > 1080 ? window.screen.height : 1080

            },
            optional: []
        }
    };

    console.log("=== window.constraintsError === " + JSON.stringify(window.constraintsError, null, 4));
    return callback(window.constraintsError, screen_constraints);
}
```

Code Extract 4-16 Broadcasting server Screen-capturing.js

The code snippet 4-16 shows how the function that get's call kurento-utils received a different send source other than webcam. This function confirms that the screen constraints are valid. Then and returns the screen constrains for the creation of peer connection with screen capture media offer. The process is then repeated as it was for the webcam streaming with peer connection setting up and pipeline with endpoints creation.

4.2.2.3 Web cam and Screenshare parallel streaming

The implementation of this component within the system required a lot of problem solving, as I encountered number of difficulties when it came to the implementation of this feature. In order to implement this feature, I was required to create two peer connections per client, negotiate them with the kurento client on the server side, this also leads to creation of two endpoints per client on the media pipeline, different sdpOffers for the two media streams had to be sent and different iceCandidates had to be sent, the two media streams had to be completely independent of each other yet negotiated simultaneously to the kurento client. This had lead to the point were I had hit a brick wall in the development process as I couldn't properly negotiate the sdpOffers and generate answers, resulting also in incorrect iceCandidates being sent. I had to take offload the development of this feature and move on and try to re-design the application, this had sparked idea of creating channels where clients could stream their video cam footage. But continuing to work on getting the parallel streams on the side I made a breakthrough where I was able to send both sdpOffers and iceCandidates simultaneously yet independent of each other and negotiate them on the server side which lead to generating answers, but answers weren't being sent simultaneously which then I had to problem solve how to get the sdpAnswers to be sent simultaneously which lead me to rewriting the call back function to add additional parameters and sent both media sdpAnswers in a single callback. On the client side after receiving both answers I simply created if statements to deal with each sdpAnswer and iceCandidate received from the server for both media types screen and cam. Having figured it out how to negotiate both streams for the presenter side I followed the same method and replicated the presenter side parallel stream negotiation for transfer on the viewer side for the purpose of receiving the parallel streams for viewing.

Education TV

During the off time when I hit a brick wall of implementing the parallel streams I contacted kurento developers through kurento google groups for some guidance but the information that they were able to give was very limiting and they basically reinstated the idea of having both media stream to be independent of each other as a single peer connection cannot support two media streams and a single webrtcendpoint cannot support two media streams as of this moment, no code was provided to me by kurento developers and there weren't example of parallel streams available at the time of development of this feature.

4.2.3 File Upload

File upload is primarily used for uploading pdf files that will be rendered within the webpage. Initially I attempted to implement file upload using a node module called multer, which is a middleware and deals with multi part form date file posts that upload files and handles them by storing locally. Multer can be used for other functions but primarily is used for file uploads. Using multer I was able to get the file to upload to a local directory within Ubuntu server, but after each upload my node.js server crashed. I spent few days attempting to debug this issue but due to not being able to figure out the issue that was caused by using multer in crashing my node server I looked for an alternative middleware. Multer is built on top of a node module called busboy which deals with parsing form data and basically simplifies the implementation.[49] Therefore, I decided to move down a layer and use busboy as I had thought that there was some issue with multer implementation that was causing the node.js with the websocket to crash.

Using express REST API and making it handle post request method and let the busboy middle ware handle the form data that had been sent, which is used for uploading files. I was able store the uploaded file within the Ubuntu local file system within crashing my server.[50]

```
app.post('/',function(req, res){
    console.dir(req.file + ' ' + req.files);

    var busboy=new Busboy({headers:req.headers});

    busboy.on('file', function(fieldname, file, filename, encoding, mimetype) {
      console.log('File [' + fieldname + ']: filename: ' + filename + ', encoding: ' + encoding + ', mimetype: ' + mimetype);

        //saving file to local directory
        var saveTo = path.join(fileDir,path.basename(filename));
        file.pipe(fs.createWriteStream(saveTo));

      file.on('data', function(data) {
        console.log('File [' + fieldname + '] got ' + data.length + ' bytes');
         });
      file.on('end', function() {
        console.log('File [' + fieldname + '] Finished');
      });
    });
    busboy.on('field', function(fieldname, val, fieldnameTruncated, valTruncated, encoding, mimetype) {
      console.log('Field [' + fieldname + ']: value: ' + inspect(val));
    });
    busboy.on('finish', function() {
       console.log('Done parsing form!');
       //Refresh file list
     var tmpfileList=fs.readdirSync(fileDir);


    fileList=tmpfileList;

      res.writeHead(303, { Connection: 'close', Location: '/' });
      res.end();
    });
    req.pipe(busboy);
});
```

Code Extract 4-17 Broadcasting Server server.js File Upload handling

The express REST API is used to handle the post request on route to index page, which is the presenter page. An instance of busboy is created, if the request header contains a file, then all the information

regarding the file can be extracted. Creating a pipe with and passing in file write stream as a callback function, writes the data to local system on the Ubuntu server. Once the new file has been update we read the uploaded file directory and store the new file list. File list array is used to for rendering the file list on the client side as it contains the path to each file in directory. The file list is created dynamically for display on the HTML page using javascript.

The functionality of switching of who is presenting had been developed using multiple messages and client session tracking. An observer client had been created that acts as a control for determine who is streaming at a particular time, due to the model allowing only a single broadcaster at a time. When a message is sent to the server by the control with the new Presenter id, the server checks if a presenter is currently streaming if the presenter is streaming then that presenter is stopped and a new presenter started depending on the new id and all the other clients are restarted to be the viewer clients.

4.2.4 PDF Rendering

PDF rendering initially was planned to be implemented using PHP.js library, but due to that library being extremely slow and outdated I went against using this library for PDF rendering. Instead of PHP.js I decided on using HTML object tag which allows for rendering the PDF using browsers own file renderer which chrome has support. The pdf files that are uploaded discussed in section 4.2.3, have a button beside each file that allows for rendering that file within the set rendering window. Clicking this button send the path of the file to the server which is then server on receiving the message containing this path for pdf send it's to all clients, client's on receiving the message from the server a function get's invoked that dynamically changes the path within the object tag for the pdf file to the new pdf and the new pdf is loaded within the window.

```
Rendering
<div id="renderContainer">
    <object id="renderer" data="./pdfs/WebRTC.pdf" type="application/pdf" width="100%" height="100%">
    <p>It appears you don't have a PDF plugin for this browser.
     No biggie... you can <a href="./pdfs/WebRTC.pdf">click here to
     download the PDF file.</a></p>
    </object>
</div>
</div>
```

Code Extract 4-18 Index.html

The code snippet shows the object html tag that is used for rendering PDF's

4.2.5 Recording

Recording of media stream's is done using kurento recorder endpoint, the recorder endpoint is created on the presenter's side of the pipeline and connected to the presenters WebRTC endpoint. Recorder endpoint records everything that goes through webrtcEndpoint into local file system, the recorder seemed easy to set up but I kept encountering issue with recordings not having audio, which lead me keep attempting to write the recording endpoint implementation as I thought the recorder endpoint implementation was at fault here I had researched similar errors but nobody had the issue I was having. After numerous hours of testing and re-coding the recorder implementation I ended up with the conclusion that the recorder end point has be to be correctly configured as it is presented a video recording, and it was not producing any errors on server side or kurento media server as I extensively checked the logs for both. I then discovered that recorder endpoint produces recordings with Opus audio codec and the player which I used (VLC) does not support opus audio codec. By testing the video with a browser the recording was played with audio, which proved that the recorder endpoint was configure appropriately. Recording of streams cannot be invoked at any time the users require so as recording of the stream starts when ever the presenter starts presenting as the recording has to be

connected to webrtcendpoint and this cannot be performed when media is going through the endpoint already.

```
421                 // Recorder endpoint for recording cam stream
422
423                 pipeline.create('RecorderEndpoint', camRecordingURI, function(error, recorderEndpointCam){
424
425                     if(error){
426                         console.log('Recording error');
427                         return callback(error);
428                     }else{
429                         console.log('Recording started');
430                     }
431
432                     presenter.recorderEndpointCam=recorderEndpointCam;
433
434
```

Code Extract 4-19 server.js recorder endpoint

Code snippet 4-19 shows the creation for recorder endpoint on the pipeline. camRecordingURI specifies the path of where the recording is to be stored and using which format kurento recording endpoint currently supports only two formats .webm and mp4 for this project .webm was used as it's easier to embed into webpages due to the format specifically created for this function and it uses less space on disk.

```
438
439                 pipeline.create('WebRtcEndpoint', function(error, webRtcEndpointCam) {
440                     if (error) {
441                         stop(sessionId);
442                         return callback(error);
443                     }
444
445                     if (presenter === null) {
446                         stop(sessionId);
447                         return callback(noPresenterMessage);
448                     }
449
450                     webRtcEndpointCam.connect(recorderEndpointCam);
451                     recorderEndpointCam.record();
452
453                     presenter.webRtcEndpointCam = webRtcEndpointCam;
454
455
456                     if (candidatesQueueCam[sessionId]) {
457                         while(candidatesQueueCam[sessionId].length) {
458                             var candidate = candidatesQueueCam[sessionId].shift();
459                             webRtcEndpointCam.addIceCandidate(candidate);
460                         }
461                     }
462
```

Code Extract 4-20 server.js connecting recorder endpoint

The code snippet 4-20 line 450 shows the connection of webRtcEndpoint to a recorderEndpont and then invocation a for a record function on the recorder endpoint, which starts recording anything going through the webrtcendpoint.

4.2.6 Chat Functionality

The chat functionality is very basic client to server messaging and displaying the messages that go through the server back to all the clients that are currently connected, the functionality of the chat wasn't extended any further due to running out of time. The flaw with this current implementation is that there is no backlog of chat so any new clients connected won't be able to see the messages that were sent prior to them connecting.

4.3 Development Environments

4.3.1 Local

Local environment was set up by installing node.js on a local system running OSX 10.13 on MacBook Pro 2015. This environment was used early in development when there was no need for client connection testing.

4.3.2 Remote

Remote development environment was set on Amazon AWS running Ubuntu 14.04 LTS server which was used for deploying the web application.

4.3.3 Sublime

Sublime is a lightweight text editor that wasn't used as a main editor for editing code during development, but it was occasionally used to open logs and editing JSON files due to it's really good search capabilities.

4.3.4 Coda

Coda is another text editor which was used as the main editor for development of the project. The reason why coda was used as a main editor as it makes it easy to access all files in relation the web application, it provides easy method of transferring files from local storage to remove server, it provides the ability to connect through ssh and viewer remote server directory. Coda also provides preview functionality for HTML pages and quick update sync when editing CSS files.

4.3.5 Terminal

OSX terminal was used for connecting to remote servers in order to gran privileges, install new modules, libraries and running node servers.

4.3.6 Git Hub

GIT hub was used for version control of the project, the project was pushed to a private repository during the development.

4.4 External API's

4.4.1 Kurento API

API for simplifying implementation of WebRTC API and is used in conjunction with Kurento Media Server.

4.4.2 Adapter.js

Adapter.js API is used to take care of the different implementation of WebRTC that is needed. This unifies the implementation of WebRTC on all browsers with the use of this API.

4.4.3 Bootstrap.js

Bootstrap API is used for HTML template development as it divides page into grids which then can be customized for any needs and it also provides a lot of styling for html page.

4.4.4 Demo-console.js

Demo console API used for displaying console log information within the webpage in order to be able to see errors that occur within the server faster.

4.4.5 Busboy.js

Busboy API is middleware for multi form data handling used for file uploads.

4.4.6 Ws.js

Ws API is a wrapper API that simplifies Websocket implementation.

4.4.7 YouTube API

YouTube API was in process of implementation of a feature where you could upload your recordings using YouTube API onto your YouTube channel, there are references of YouTube API within the project but the complete functionality hadn't been developed.

4.5 Cloud Deployment

4.5.1 Amazon AWS

Amazon AWS cloud services EC2 instance is used to host Ubuntu 14.04LTS server, which is used for broadcasting server. By hosting the broadcasting server on EC2 instance it takes care of load balancing and scaling due to number of client's being connected to a single broadcasting channel. This takes care of bandwidth issues due to the fact if I was to development a room functionality on a single channel this would cause severe bandwidth load to the server as the clients would be connecting to multiple rooms. By offloading each channel to a different Amazon AWS server I save on the bandwidth and the scaling is done by the EC2 instance.

For the project a single channel was created, but if there would be requirement for another channel, the user would create a copy of the existing Amazon AWS server and host a copy of broadcasting server in order to produce another channel.

4.5.2 Azure

Azure is used as a cloud service for hosting Ubuntu 14.04 LTS server and are used for deploying the channel server due to channel server not requiring large number of open ports azure was a good option to use and is much faster to set up than amazon AWS which saves valuable time.

5 System Validation

5.1 Testing

Testing was executed with multiple users for most part minimum of two users had been used for testing, after each iteration of developed feature. I will use multiple testing methods in order to get insight on usability of the system, performance and if it meets the requirements of functionality for that particular iteration as I will be following agile development methodology. The people that were involved in testing each of my iterations functionality are a few of my friends. I needed more than one client to test the functionality of my project because it's a multi client web application I was required to test it with multiple clients that's why I involved my friends in the testing process to simulate other clients. Due to my friends not being available at all times I performed tests using another tab to simulate another client or when I had accessed to another machine at home which was a better representation of the test due to the other machine using different hardware than the machine I had been developing on.

Unit testing was performed on scripts that are written for the web application. In order to unit test JavaScript, I will use QUnit.js, QUnit.js is a javascript API that provides the tools to test your JavaScript functionality and try break it. The use of QUnit.js will allow me to find bugs in my client side JavaScript code and debug them after each iteration of my development cycle. Due to development falling behind by a lot QUnit was used in early weeks of development cycle of the project, but due to falling behind in development process I had to stop writing tests because I needed the time to push forward development and try close the gap with initial plan. Same issue occurrence with Mocha js that was to test the server code early on as with QUnit.

Testing of Compatibility and functionality of features, most of feature testing had been performed on OSX 10.13 using chrome browser. Other machines that had been used for testing the compatibility of application were a desktop pc running windows 10, using chrome browser. All the tests were performed on chrome browser due to the application being developed specifically for the chrome browser. Tests for mobile was done on Samsung S4 running android 5.0.1. The largest number of clients that were attempted to have a presenter viewing was 5 this was testing in one of the Labs at DIT, using lab machines, the lab computes that this was tested on ran windows 7. Most of the presenting testing was done on the OSX as 10.13 due to having the screen capture extension installed on the system.

During the testing and debugging phases the logs had too be extensively monitored for any error occurrences. As these contained all the information on all of the connections that are being made and any of the projects components failing.

While testing recording functionality, the largest file recorded for the purpose of testing was 8 minutes long which came at about 30mbps of size. The file came out at an average of 480p resolution which isn't the best but could improve over the improvements of MCU.

Education TV

White box testing was performed for file uploads as it was the only appropriate functionality within the project were White box testing had made sense. White box testing was performed using chrome extension ARC which allows for creating file post request to the server.

Graphical user wasn't tested due to no final version had been released and due to running out of time there was no time to perform intensive usability tests and client feedback tests.

6 Project Plan

6.1 Analysis

This is a compiled list of all the dates that are for delivering features of the project.

TASK	START DATE	DURATION	END DATE
RESEARCH 1	02-Nov	7	09-Nov
INTERIM REPORT 1	10-Nov	7	17-Nov
RESEARCH 2	18-Nov	5	23-Nov
INTERIM REPORT 2	24-Nov	6	30-Nov
INTERIM REPORT 3	01-Dec	6	07-Dec
VIDEO CAM BROADCAST	08-Dec	12	20-Dec
TESTING/DEBUGGING	21-Dec	3	24-Dec
SCREEN SHARE BROADCAST	24-Dec	12	05-Jan
TESTING/DEBUGGING	06/01/2016	3	09/01/2016
FILE UPLOAD	10/01/2016	5	15/01/2016
TESTING/DEBUGGING	16/01/2016	3	19/01/2016
FILE RENDERING	20/01/2016	4	24/01/2016
TESTING/DEBUGGING	25/01/2016	2	27/01/2016
GROUP CHAT	28/01/2016	5	02/02/2016
TESTING/DEBUGGING	03/02/2016	3	06/02/2016
USER DATABASE	07/02/2016	1	08/02/2016
REGISTRATION/LOGIN	09/02/2016	1	10/02/2016
CONNECTED USER DISPLAY	11/02/2016	3	14/02/2016
TESTING/DEBUGGING	15/02/2016	3	18/02/2016
GUI HIGH LEVEL PROTOTYPE	19/02/2016	3	22/02/2016
MOBILE ANDROID UI SCALING	22/02/2016	2	24/02/2016
TALET ANDROIND UI SCALING	24/02/2016	2	26/02/2016
TESTING/DEBUGGING	27/02/2016	2	29/02/2016
GUI FINALISATION	29/02/2016	4	04/03/2016
EXTRA TASK	05/03/2016	5	10/03/2016
EXTRA TASK	10/03/2016	5	15/03/2016
EXTRA TASK	15/03/2016	5	20/03/2016
EXTRA TASK	20/03/2016	5	25/03/2016
EXTRA TASK	30/03/2016	0	30/03/2016
EXTRA TASK	30/03/2016	6	05/04/2016

Figure 6-1 Development Dates

The development process starts 08-Dec for the project and the expected completion of the project is 04/03/16. Due to not having the exact date for submission I wanted to have the finish day as early as possible in March, in order to have few weeks extra if needed some features might take longer to develop than expected and forcing my finishing date further into March.

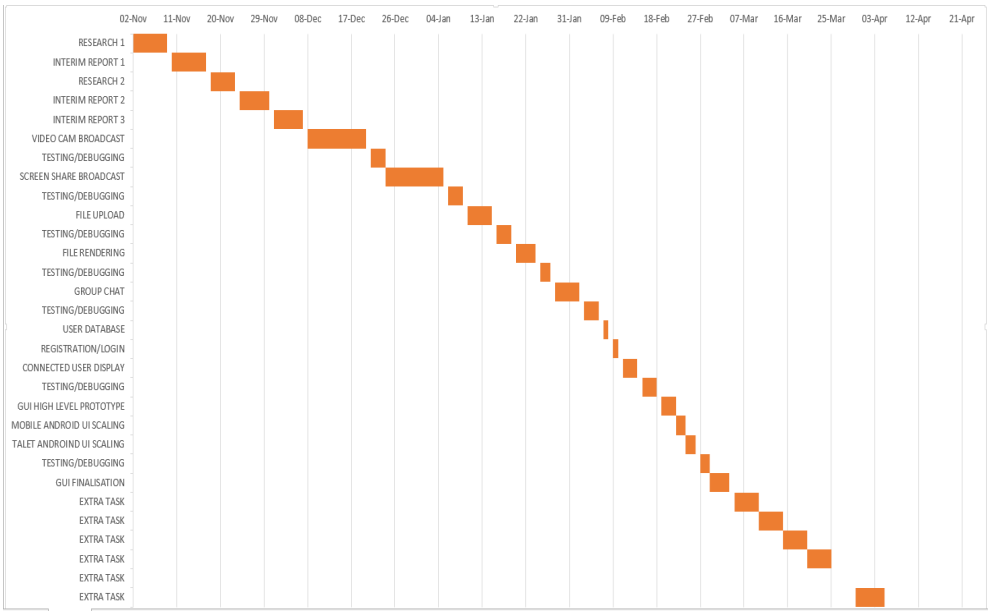


Figure 6-2 High Level Gantt Chart

From the Gantt chart we can see that the first few features have a long development period, this is due to the fact that these 2 features are of high risk therefore they need extended development process. Once the the 2 main features are complete the development process for the rest of the features is much shorter due to the fact that they are easier to develop.

Agile development Gantt chart:
The Gantt chart below displays how I utilize the agile development methodology in development of my project.

Task Name	Start Date	End Date	Duration	Predecessors	% Complete	Assigned To	Comments
Sprint 1 Video Cam Broadcast	**12/08/15**	**12/23/15**	**12d**		**0%**		
Requirements Analysis	12/08/15	12/09/15	2d		0%		
Research	12/10/15	12/11/15	2d	2	0%		
Design	12/12/15	12/12/15	1d		0%		
Coding	12/13/15	12/23/15	9d		0%		
Unit Testing	12/23/15	12/23/15	1d		0%		
Sprint 2 Screen Share Broadcast	**12/24/15**	**01/08/16**	**12d**		**0%**		
Requirements Analysis	12/24/15	12/25/15	2d		0%		
Research	12/26/15	12/28/15	2d		0%		
Design	12/29/15	12/29/15	1d	9	0%		
Coding	12/31/15	01/07/16	6d		0%		
Unit Testing	01/08/16	01/08/16	1d		0%		
Sprint 3 File Upload	**01/09/16**	**01/27/16**	**14d**		**0%**		
Requirements	01/09/16	01/09/16	1d		0%		
Research	01/10/16	01/10/16	1d		0%		
Design	01/11/16	01/11/16	1d		0%		
Coding	01/11/16	01/18/16	6d		0%		
Unit Testing	01/18/16	01/18/16	1d		0%		
File Render	**01/19/16**	**01/27/16**	**7d**		**0%**		
Requirements	01/19/16	01/19/16	1d		0%		
Research	01/20/16	01/20/16	1d		0%		
Design	01/21/16	01/21/16	1d		0%		
Coding	01/21/16	01/27/16	5d		0%		
Unit Testing	01/27/16	01/27/16	1d		0%		
Sprint 4 Group Chat	**01/28/16**	**02/06/16**	**7d**		**0%**		
Requirements Analysis	01/28/16	01/28/16	1d		0%		
Research	01/29/16	01/29/16	1d		0%		
Design	01/31/16	01/31/16	1d		0%		
Coding	01/31/16	02/05/16	6d		0%		
Unit Testing	02/06/16	02/06/16	1d		0%		
Sprint 5 User Login	**02/07/16**	**02/18/16**	**10d**		**0%**		
Requirements	02/07/16	02/07/16	1d		0%		
Research	02/08/16	02/08/16	1d		0%		
Design	02/09/16	02/09/16	1d		0%		
Database	02/10/16	02/10/16	1d		0%		
Coding	02/10/16	02/10/16	1d		0%		
Registration	02/11/16	02/12/16	2d		0%		
Coding	02/11/16	02/12/16	2d		0%		
Login Authentication	02/12/16	02/15/16	2d		0%		

Figure 6-3 Gantt Chart Dates

Coding	02/12/16	02/15/16	2d		0%		
Connected User Display	02/15/16	02/16/16	2d		0%		
Coding	02/15/16	02/16/16	2d		0%		
Unit Testing	02/18/16	02/18/16	1d		0%		
Sprint 6 GUI High level Prototype	**02/19/16**	**02/22/16**	**2d**		**0%**		
Requirements	02/19/16	02/19/16	1d		0%		
Research	02/19/16	02/19/16	1d		0%		
Design	02/20/16	02/20/16	1d		0%		
Coding	02/20/16	02/22/16	2d		0%		
Sprint 7 Mobile UI Scaling	**02/22/16**	**02/24/16**	**3d**		**0%**		
Requirements	02/22/16	02/22/16	1d		0%		
Research	02/23/16	02/23/16	1d		0%		
Design	02/23/16	02/23/16	1d		0%		
Coding	02/23/16	02/24/16	2d		0%		
Sprint 8 Tablet UI Scaling	**02/24/16**	**02/26/16**	**3d**		**0%**		
Recuirements	02/24/16	02/24/16	1d		0%		
Research	02/25/16	02/25/16	1d		0%		
Design	02/25/16	02/25/16	1d		0%		
Coding	02/25/16	02/26/16	2d		0%		
Sprint 7 GUI Finalisation	**02/27/16**	**03/04/16**	**6d**		**0%**		
Testing	02/27/16	02/29/16	2d		0%		
Coding	03/01/16	03/04/16	4d		0%		

Figure 6-4 Gantt Chart Dates continued

Education TV

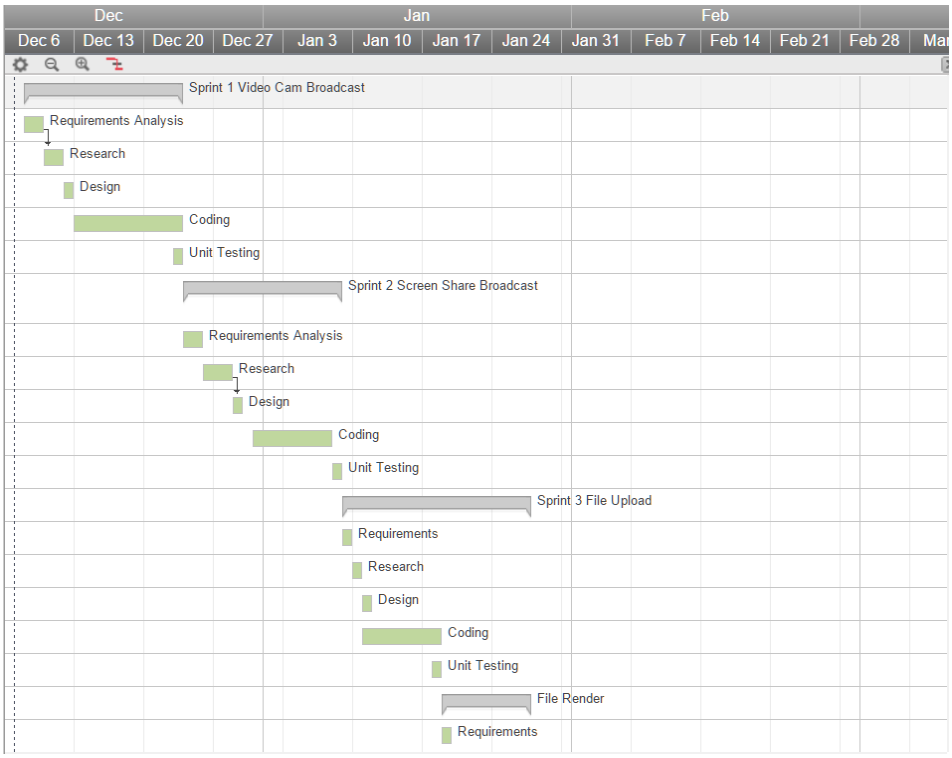


Figure 6-5 Gantt Chart Graphic

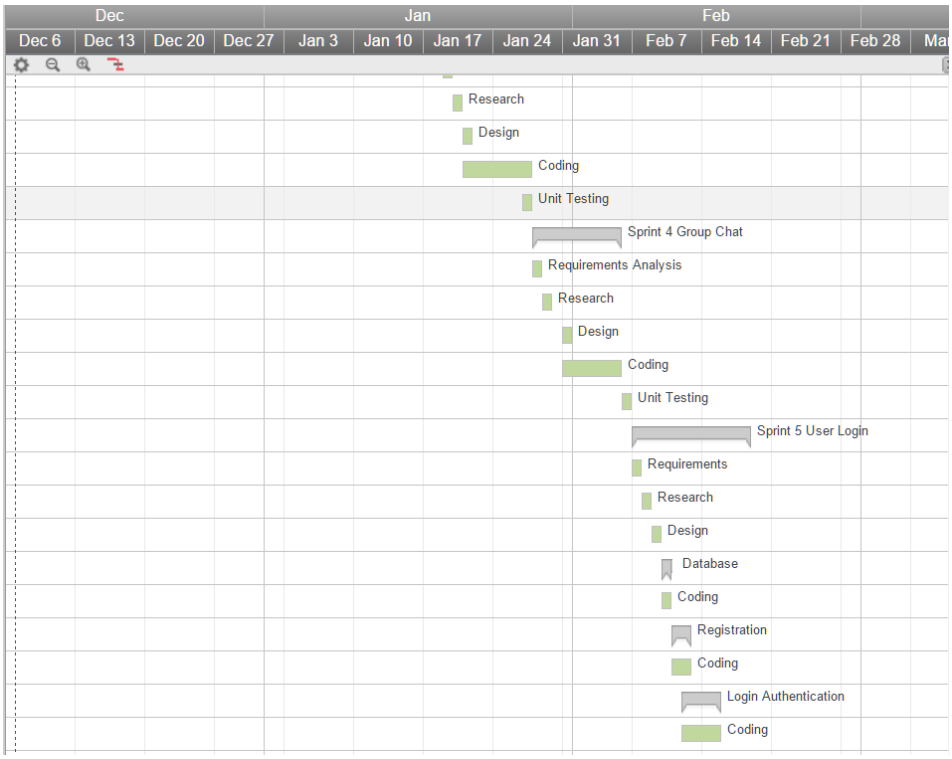


Figure 6-6 Gantt Chart Graphic Continued

Education TV

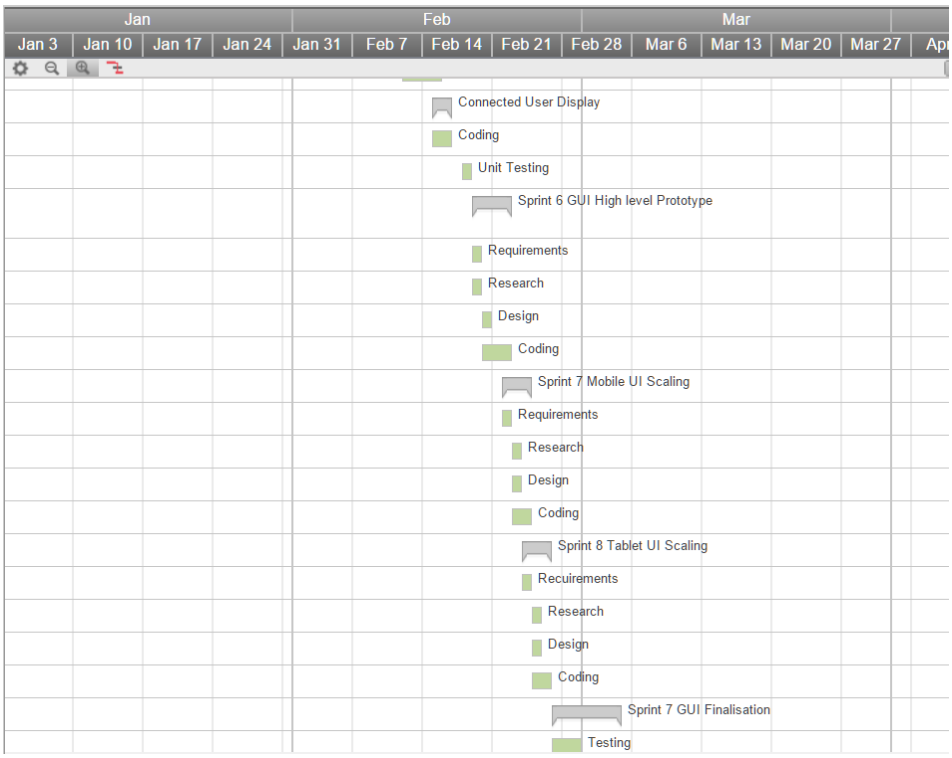


Figure 6-7 Gantt Chart Graphic Continued

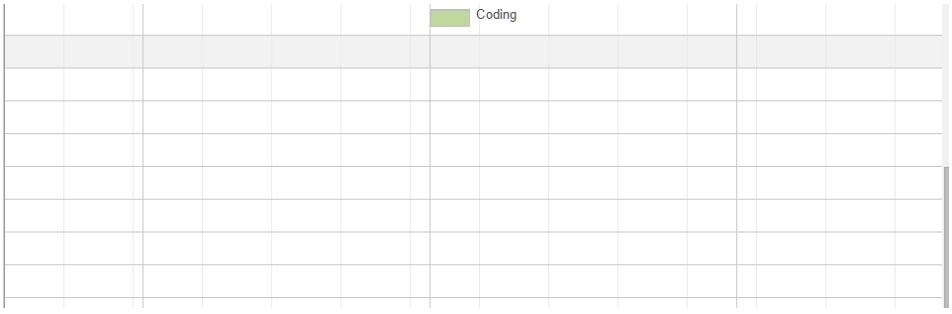


Figure 6-8 Gantt Chart Graphic End

Each sprint iteration starts with requirements analysis for that particular feature that is going to be developed, then we perform research based on the requirements and formulate a design. Once we have the design we start the coding process and when the coding process is finished we begin unit testing of all the code.

A lot of thought went into project plan within Gantt chart what seemed a good estimate of time required for each feature and component of the project to be implemented. In the review section 6.2 will be an explanation whether this method of planning worked.

6.2 Review

The initial plan was a good guideline of where I was supposed to be during each phase of the development for each feature, due to unforeseen circumstances the development ended up behind due to encountering severe issues with the web server working with Kurento(MCU) server early on in the development which delayed further implementation of the project as getting the server with MCU working on amazon AWS was the main hosting service that was available. Following the server getting

to work there were multiple issues encountered when it came to get the screen share to work for the project, I didn't anticipate the problems that I would encounter during the development of this feature, during this process I had to use already developed Screen Capturing extension that was open sourced and modify capturing script to suit my needs, but I ended up wasting a lot of time prior to using this extension in attempts to develop my own functional extension which was a failure. Another major development issue was encountered when in attempt to stream both the webcam and screen share simultaneously this feature took much longer than originally anticipated as I kept encountering many errors and debugging took enormous amount of time this was due to no previous examples to research as there were no implementations online of parallel streaming. The early struggle with the implementation had delayed the other features implementation and I wasn't able to catch up to the initial Gantt chart plan ending up not being able to complete all of the functionality and features I initially had planned.

The plan had other flaws such as not incorporating January examinations which went on for 4 days in January and during this time there was the need for a lot of study which took time away from the development of the project. The plan didn't take any considerations for assignments and lab tests that were given in both semesters, as on those weeks the amount of time dedicated to the development of the project had been reduced, which combined with other shortfalls previously discussed made a wider gap between the development according to the plan and the development stage I was in.

6.3 Evaluation

Evaluating the progress made according to the initial plan if the project had been repeated. I would take into consideration all of the assignment dates, lab tests and semester one exams when forming the development plan, this would already give me a better estimate of average time that I would be able to work on the project on certain weeks that have exams, assignments or lab tests. The development process started way to late and the development of the project had been started earlier in semester one. There was a lot of research done on WebRTC and how it is implement on peer to peer basis and examples that had been run locally to get a feeling for how it worked, but due to spending a large amount of research on WebRTC I left most of the Kurento(MCU) research for when the development started, using the Kurento(MCU) which uses it's own libraries for implementation of WebRTC communication with peers and MCU meant that I had to research WebRTC implementation using adapter.js library and kurento-utils.js library in order to start implementation of my project, which made the previous large amount of time spent on researching WebRTC have little value, due to this research had been carried out at early stage of development rather than when I had been researching WebRTC and working on it's examples it really slowed me down in progressing with the development. If I were to to repeat I would have researched Kurento(MCU) more extensively prior to development as it is at the core of the project and it's dealing with the streams that are being streamed to clients.

Deploying the application to Amazon AWS EC2 Ubuntu server should of had been performed much earlier as this required a lot of setting up in order to get the webserver working and functional and there were numerous issues being encountered in the processes.

Due to limited time frame for the project and number of issues encountered during the process of the project, functionalities and features had to been cut from the project that were in the initial design, registration, login and authentication had to be cut from the project, UI for selecting the which client is presenting at time, voice call functionality between viewer client and presenter. Improvements to

GUI and securing the application were also cut from the project due to lack of time. This severely made the application suffer from usability and functionality.

7 Conclusion

In conclusion the project idea had evolved extensively from initial proposal, but still having its core idea implemented, there were a number of difficulties and challenges encountered throughout the development of the project, which led trough a number of redesigns during development. Even thought initial design hadn't been achieved a form of initial proposal had been implemented with the current redesign of the project. The requirement for early redesigns were needed due to encountering number of issues early in the development process and throughout the development which caused re-evaluation of what can be implemented for this project feature wise and which use cases it could support from initial proposal and which required a redesign or some aspects of the initial proposal could not be implemented therefore requiring a redesign. Overall the project working on this project had provided me with lot's of experience in different aspects of software development.

What I had succeeded feature wise for my project, is to develop a system with the core features initially planned that make use of WebRTC and Kurento (MCU) and additional feature such as recording that was not part of initial implementation design. The project doesn't work as a complete systems and there is lack of security and authentication for different users. The project isn't as accessible in terms of usability as initially planned due to it working for a single type of browser which is chrome and even though chrome has 69% usage in 2016(53) from all available browsers, but if the project could support Firefox browser which has the usage of 18.6% between both browsers usage would reach 87.6% and as WebRTC is supported by Firefox this would allow for more users able to use my project.[45] Due to WebRTC requiring a different implementation in order to function for Firefox browser I wasn't able support both browsers due to lack of development time for this project. The project still lacks many features compared to other educational web applications and there are many improvements to be made in order for it to deliver a true rich environment for education through the web.

The learning outcomes from this project that I had acquired, I gained were the ability to work with newest technologies that are still in development. Due to extensive research on three tier architectures and MCU, I gained a lot of experience in working with three tier architectures for client server applications. I gained experience working with MCU and Node.js servers. When it came to non technical side of learning outcomes, I gained thorough experience in project planning, usability testing and documentation. A wide variety of skills had been acquired throughout the research and development of the project both technical and non technical.

8 Bibliography

[1] R. Lyons, "The Daft.ie Rental Report", www.daft.ie, 2016. [Online]. Available: http://www.daft.ie/report/q3-2015-daft-rental-report.pdf. [Accessed: 07- Dec- 2015].

[2]"Public transport fares to rise for many before Christmas", The Irish Times, 2015. [Online]. Available: http://www.irishtimes.com/news/ireland/irish-news/public-transport-fares-to-rise-for-many-before-christmas-1.2411425. [Accessed: 07- Dec- 2015].

[3]"BigBlueButton:Architecture", Docs.bigbluebutton.org. [Online].Available: http://docs.bigbluebutton.org/1.0/10architecture.html. [Accessed: 07- Dec- 2015].

[4]"Red5 Media Server", Red5.org. [Online]. Available: http://red5.org/. [Accessed: 07- Dec- 2015].

[5]"BigBlueButton : HTML5 Overview", Docs.bigbluebutton.org. [Online]. Available: http://docs.bigbluebutton.org/labs/html5-overview.html. [Accessed: 07- Dec- 2015].

[6]"FAQ", Webex.com, 2015. [Online]. Available: http://www.webex.com/faqs.html. [Accessed: 07- Dec- 2015].

[7]"Cisco WebEx — the global leader in video conferencing.", Webex.com. [Online]. Available: http://www.webex.com/why-webex/overview.html. [Accessed: 07- Dec- 2015].

[8]"Cisco Validated Design Guide, Cisco WebEx Social Release 3.3 - Architecture Overview [Cisco WebEx Social]", Cisco. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/collaboration/ecp/cvd/3_3/cvd_33/arch_overview.html. [Accessed: 07- Dec- 2015].

[9] J. Roettgers, "Gigaom | The technology behind Google+ Hangouts", Gigaom.com, 2011. [Online]. Available: https://gigaom.com/2011/06/30/google-hangouts-technology/. [Accessed: 07- Dec- 2015].

[10]"Get started with Hangouts - Computer - Hangouts Help", Support.google.com. [Online]. Available: https://support.google.com/hangouts/answer/2944865?hl=en. [Accessed: 07- Dec- 2015].

[11] Google+, "Google Finally Kills Off GoogleTalk and XMPP (Jabber) Integration", Disruptive Telephony. [Online]. Available: http://www.disruptivetelephony.com/2015/02/google-finally-kills-off-googletalk-and-xmpp-jabber-integration.html. [Accessed: 07- Dec- 2015].

[12]J. Roettgers, "Gigaom | The technology behind Google+ Hangouts", Gigaom.com, 2011. [Online]. Available: https://gigaom.com/2011/06/30/google-hangouts-technology. [Accessed: Mar- 2016].

[13]"VoIP: Skype architecture", Slideshare.net. [Online]. Available: http://www.slideshare.net/Sandra4211/voip-skype-architecture. [Accessed: 07- Dec- 2015].

Education TV

[14]"About Skype - What is Skype", Skype.com. [Online]. Available: http://www.skype.com/en/about/. [Accessed: 07- Dec- 2015].

[15]S. A. Baset and H. G. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol". [Online]. Available: http://www1.cs.columbia.edu/~salman/publications/skype1_4.pdf. [Accessed: 07- Dec- 2015].

[16] [Online]. Available: http://www.webrtc.org/home. [Accessed: 07- Dec- 2015].

[17]"Architecture|WebRTC",Webrtc.org. [Online]. Available: http://www.webrtc.org/architecture. [Accessed: 07- Dec- 2015].

[18]"HTML5 Introduction", W3schools.com. [Online]. Available: http://www.w3schools.com/html/html5_intro.asp. [Accessed: 07- Dec- 2015].

[19]"1 Introduction — HTML5", W3.org. [Online]. Available: http://www.w3.org/TR/html5/introduction.html#background. [Accessed: 07- Dec- 2015].

[20]N. Foundation, "About | Node.js", Nodejs.org. [Online]. Available: https://nodejs.org/en/about/. [Accessed: 07- Dec- 2015].

[21]"MEAN.IO - MongoDB, Express, Angularjs Node.js powered fullstack web framework - MEAN.IO - MongoDB, Express, Angularjs Node.js powered fullstack web framework", MEAN.IO. [Online]. Available: http://mean.io/#!/. [Accessed: 07- Dec- 2015].

[22]"LAMP Stack", Turnkeylinux.org. [Online]. Available: https://www.turnkeylinux.org/lampstack. [Accessed: 07- Dec- 2015].

[23] [Online]. Available: https://aws.amazon.com/ec2/. [Accessed: 07- Dec- 2015].

[24] [Online]. Available: https://aws.amazon.com/documentation/elastic-beanstalk/. [Accessed: 07- Dec- 2015].

[25]"Table of Contents — Kurento 6.4.0 documentation", Kurento.org. [Online]. Available: http://www.kurento.org/docs/6.1.0/what_is_kurento. [Accessed: 07- Dec- 2015].

[26]"Table of Contents — Kurento 6.4.0 documentation", Kurento.org. [Online]. Available: http://www.kurento.org/docs/6.1.0/introducing_kurento.html. [Accessed: 07- Dec- 2015].

[27]"Red5 Media Server", Red5.org. [Online]. Available: http://red5.org/. [Accessed: 07- Dec- 2015].

[28]"Red5 Pro", Red5pro.com. [Online]. Available: https://red5pro.com/#price. [Accessed: 07- Dec- 2015].

[29]"About HTML5 WebSocket - Powered by Kaazing", Websocket.org. [Online]. Available: http://www.websocket.org/aboutwebsocket.html. [Accessed: 07- Dec- 2015].

Education TV

[30]"What is Meteor.js?", Joshowens.me, 2014. [Online]. Available: http://joshowens.me/what-is-meteor-js/. [Accessed: 07- Dec- 2015].

[31]"AngularJS — Superheroic JavaScript MVW Framework", Angularjs.org. [Online]. Available: https://angularjs.org/. [Accessed: 07- Dec- 2015].

[32] C. McKeachie, "React.js and How Does It Fit In With Everything Else? - Funny Ant", Funny Ant, 2014. [Online]. Available: http://www.funnyant.com/reactjs-what-is-it/. [Accessed: 07- Dec- 2015].

[33]"CoffeeScript", Coffeescript.org. [Online]. Available: http://coffeescript.org/. [Accessed: 07- Dec- 2015].

[34] A. O. Di Benedetto, "Does Technology Influence Teaching Practices in the Classroom?", 2005. [Online]. Available: https://www.stcloudstate.edu/tpi/initiative/documents/technology/Does%20Technology%20Influence%20Teaching%20Practices%20in%20the%20Classroom.pdf. [Accessed: 07- Dec- 2015].

[35]"Pack your tablets: Irish schools ditch the textbooks to go digital", The Irish Times, 2014. [Online]. Available: http://www.irishtimes.com/news/education/pack-your-tablets-irish-schools-ditch-the-textbooks-to-go-digital-1.2017467. [Accessed: 07- Dec- 2015].

[37]"Semantic Scholar", Semanticscholar.org. [Online]. Available: https://www.semanticscholar.org/paper/Voice-over-IP-Varshney-Snow/0110436c4cf0f7ce5c410d8fd3553f07da74ebd1. [Accessed: 07- Dec- 2015].

[38]"How VoIP Works", HowStuffWorks, 2001. [Online]. Available: http://computer.howstuffworks.com/ip-telephony.htm. [Accessed: 07- Dec- 2015].

[39]"draft-ietf-rtcweb-security-08 - Security Considerations for WebRTC", Tools.ietf.org. [Online]. Available: https://tools.ietf.org/html/draft-ietf-rtcweb-security-08#section-3.2. [Accessed: 07- Dec- 2015].

[40]L. Slattery, "The impact of Google's new Chrome security policy on WebRTC | TokBox Blog", Tokbox.com, 2015. [Online]. Available: http://www.tokbox.com/blog/the-impact-of-googles-new-chrome-security-policy-on-webrtc/. [Accessed: 07- Dec- 2015].

[41]"Secure your site with HTTPS - Search Console Help", Support.google.com. [Online]. Available: https://support.google.com/webmasters/answer/6073543?hl=en. [Accessed: 07- Dec- 2015].

[42]G. Conklin, G. Greenbaum, K. Lillevold, A. Lippman and Y. Reznik, "Video coding for streaming media delivery on the Internet", IEEE Trans. Circuits Syst. Video Technol., vol. 11, no. 3, pp. 269-281, 2001.

[43]C. Zhang and J. Liu, "On Crowdsourced Interactive Live Streaming: A Twitch.TV-Based Measurement Study". [Online]. Available: http://arxiv.org/pdf/1502.04666.pdf. [Accessed: 07- Dec- 2015].

Education TV

[44]A. Fecheyr-Lippens, "A Review of HTTP Live Streaming", 2010. [Online]. Available: http://files.andrewsblog.org/http_live_streaming.pdf. [Accessed: 07- Dec- 2015].

[45]"Browser Statistics", W3schools.com. [Online]. Available: http://www.w3schools.com/browsers/browsers_stats.asp. [Accessed: Mar- 2016].

[46]"Kurento/kurento-tutorial-node", GitHub. [Online]. Available: https://github.com/Kurento/kurento-tutorial-node/tree/master/kurento-one2many-call. [Accessed: Mar- 2016].

[47]"muaz-khan/WebRTC-Experiment", GitHub. [Online]. Available: https://github.com/muaz-khan/WebRTC-Experiment/tree/master/Chrome-Extensions/desktopCapture. [Accessed: Mar- 2016].

[48]"muaz-khan/Chrome-Extensions", GitHub, 2014. [Online]. Available: https://github.com/muaz-khan/Chrome-Extensions/tree/master/Screen-Capturing.js. [Accessed: Mar- 2016].

[49]"expressjs/multer", GitHub, 2015. [Online]. Available: https://github.com/expressjs/multer. [Accessed: Mar- 2016].

[50]"mscdex/busboy", GitHub. [Online]. Available: https://github.com/mscdex/busboy. [Accessed: Mar- 2016].