Nikita Korhonen

Television Director Simulator

Multi-camera Streaming application

Helsinki Metropolia University of Applied Sciences

Information Technology

Bachelor of Engineering

Thesis

3 April 2017

Author(s) Title Number of Pages Date	Nikita Korhonen Television Director Simulator 43 pages + 0 appendices 3 April 2017
Degree	Information Technology
Degree Programme	Bachelor of Engineering
Specialisation option	Software Engineering
Instructor(s)	Olli Alm senior lecturer, Pekka Korvenoja senior lecturer

This thesis is about a television director simulator, which is an application which aims to provide assistance in training the live multi-camera directors. The project requirement was to build an application that could be used for providing multiple students at the same time with a possibility to experience a multi-camera environment that live broadcasting events have. The project goal arose from the need to improve the education methods for teaching multi-camera directors.

The thesis looks into different multi-camera streaming concepts, explains their application purposes, capabilities and drawbacks and assesses the usefulness for the television director simulator. There were many design challenges for the simulator, but they were successfully resolved. The solutions are explained in detail in the Implementation section of the thesis. The television director simulator ended up to be a MEAN stack web-application, with Nginx acting as a proxy server to the application and FFmpeg on the backend to manipulate the video files.

In the end, the application that was built as a result of the project allows to speed up the education process of television directors. The application is unique and thus it has a potential on the market. Students can use the application at home at any time they want to practise the television directors work through a simulated environment.

Keywords	Television director, multi-camera setup, video streaming, MEAN stack, Node.js, MongoDB, Express, HTML5, JavaScript

Contents

1 Introduction

Nowadays there are many talk shows, sports programs, theatrical plays and other similar types of shows that are broadcasted live through the world wide web or on a television. These broadcasts have usually multiple cameras shooting the event, but only one at a time is shown to the public. Therefore, there is a special person, called television director, dedicated to deciding on what camera to show to the public at any given time. Television directors' responsibilities might differ depending on whether the production is live or a recording. The thesis is about an application that could simulate the multi-camera environment in order for students to practice television directors live directing tasks. The goal of the project was to build a simulator so that students in Metropolia University of Applied Sciences could use it in their studies.

The thesis covers the background of multi-camera streaming setups, applications' development process and results. I will first explain the reasons behind the simulator and go through trends in a multi-camera streaming environment, explain and assess their setups and application purposes. After that I will look into different video streaming methods and technologies that currently exist. Finally, I will explain what design challenges we faced, how we approached and solved them. Since there is always room for improvement, I will also mention possible future improvements and extra features that the application could benefit from.

As for the motivation, the topic of video streaming got me interested since I had never developed anything that uses video streaming, while I am dealing with it on a daily basis as a client - youtube, twitch.tv and skype, to name a few technologies that use video streaming. I was also very interested in how live broadcasts deal with the latency and synchronization issues. I saw a perfect opportunity for gaining knowledge on the topic from this project and learning about the implementation of video streaming services.

Senior lecturer Pekka Korvenoja made the request for such a simulator and defined the necessary requirements for the application. The project was supervised by senior lecturer Olli Alm, while the application itself was done by me and my partner Oleg Batalin. The project lasted for six months and during that time we had to come up with an applicable solution for a multi-camera director simulator.

2 Concepts and technology of director simulator

A director simulator will have to handle many cameras, as a student will have to choose between them. There are many multi-camera streaming setups and the thesis covers only those relevant to the topic. In order to understand what the television director simulator is or should be, there are two issues that should be examined: reasons behind the simulator and the technology which can determine the properties and capabilities of the director simulator.

2.1 Reasons behind the simulator

Metropolia University of Applied Sciences has a course dedicated to training television directors. They have a studio where they can stage up a talk-show or a play, put a student behind each of the four cameras that they have and other students in the operator room - one director and 2-3 operators that would listen to the director's commands. One student manages graphical effects on the screen, like subtitles, advertisement and/or inserts, the second one switches between cameras and the last one manages sounds. Such a setup has many issues with training television directors:

- Extensive training sessions.
- Exhausting experience.
- Pre-scheduled training sessions.
- Short storing period for results.

The conventional approach that the media-course has is that the stage up or the "play" of the talk show needs to be repeated for each student separately. This can be anywhere from 15 to 20 times or maybe even more, and one play can be anywhere from 8 to 15 minutes plus the setup. Furthermore, the teacher should give feedback to each student about their performance, so the first thing they might do after a play is to watch it and learn what mistakes were made during the demonstration of the play, in order for the remaining students to avoid the same mistakes. This in its own turn almost doubles the time required to complete such a training session.

Another thing is that setting up such a play 15 or more times can be very exhausting for the participants. Therefore these “plays” have to be separated into sessions and even then not every student might get to experience the multi-camera directors’ work. Furthermore, the quality of a play may vary from session to session, which gives each student somewhat different material to work with. This might seem good at first, as each student has his or her own, to some extent unique, task to deal with. However, the quality of the play over repetitions can get so low that in the end choosing between the cameras might prove to be meaningless as all of them might not be fit in to the criteria of broadcasting and thus the student is left with a bad task and experience.

Even in a scenario where staging up “plays” could be avoided and instead student could work with recordings behind the operator console, it would still mean that there has to be a queue for the operator console (since there is only one in the studio). Furthermore, students most probably would not be able to save their own results for a long period of time, since keeping all the files would deplete the server storage space quite fast and thus after each course, the majority of the recordings have to be deleted or the storage space has to be extended, and the latter costs money.

The problems that arose from such a setup in the multimedia course gave birth to an idea of creating a multi-camera director simulator. The requirement was that students could use the application independently and at the same time produce their own results from the input videos. Students could potentially sit at their home computers and do the multi-camera directing assignments that the teacher issued them. Assignments, on the other hand, should come in the form of video files, containing the recordings from the “play” or a small talk-show.

2.2 Multi-camera applications

In order to understand what we would need to build a television director simulator application, we had to look into the existing multi-camera setups and streaming technologies. The multi-camera concept is applicable in a variety of fields and the setup often depends on the purpose of the application.

2.2.1 Television studio

First we looked into the television studio setup, for which the Metropolia course was training the students. Pekka Korvenoja gave us various information on this topic, since he has extensive experience in television broadcasting. The setup consists of a scene, video cameras and audio devices that capture the event on the scene. The scene can be a sports game in a big stadium or a scripted talk show in a closed space. The captured scene is then monitored and controlled from the production control room, which is located separately from the scene. From the production control room the television director and multiple operators produce the end result of a single-view video that the client sees.

Figure 1 is presents an example of a production control room with multiple cameras transmitting the event, while the main screen is showing the graphical introduction of a show with its name on it. The television director decides when to show which input stream and tells the instructions to the operators. Operators then switch the camera views and manage different kinds of inserts (short video recordings used for demonstration).

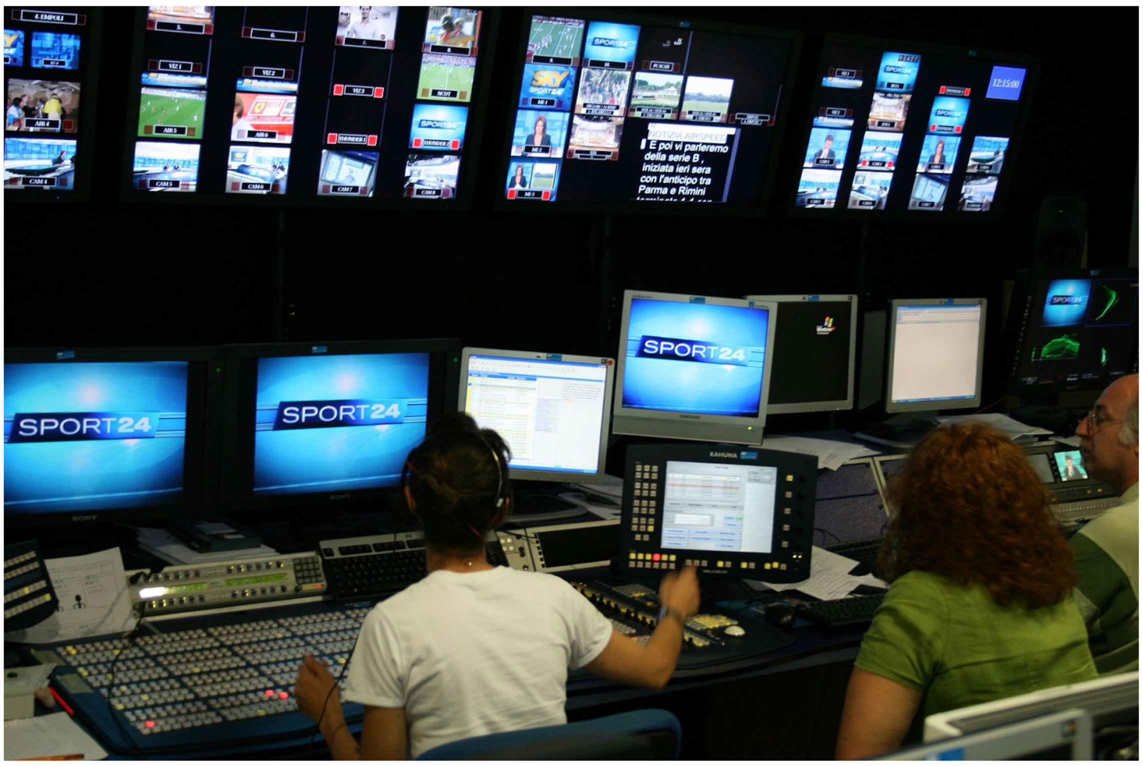


Figure 1. Example of production control room at SKY Sport24, PCR [7].

Apart from video streams, an additional operator manages sounds by monitoring the volume levels and playing musical sound effects now and then. Furthermore the quality of the video streams is also monitored for such things as contrast and lighting exposures.

The operator console consists of buttons for each of the inputs. When the button for certain input is pressed that input stream will then be shown on a preview screen. The preview screen is as big as the record screen and is used for assessing the subjective quality of a video frame. In order to move the preview input to the record screen, the operator has to trigger a “take” switch or button. Furthermore, the setup usually has also a clock, which indicates how much time is left for finishing the stream.

From a technological perspective, even though the production control room is separate from the scene, in the case of live events the two are usually located in the same area of event, in order to allow low-delay between video and audio synchronization and delivery of the stream. The camera and audio devices are directly connected to the production control room, where the synchronization happens. Modern stadiums are equipped with this type of technology in order to allow broadcasting of the events [1]. In the case of talk shows or similar type of indoor shows and plays, the event is either recorded or live broadcasted from a studio. The studio has the cameras, audio devices and the production control room all set up.

From the television director’s perspective this type of environment is considered to be offline, since the director has to be present at the production control room. In other words, there is no application, which he could use in order to remotely orchestrate the work of operators. However, in order to produce a viable television director simulator we would need to imitate in our application the production control room, with multiple cameras, a keyboard for controlling the cameras and the take button.

Furthermore, the television director’s duties are not limited to deciding on which camera to show when and the decision making often requires preparation. The television director has to prepare for broadcasting or recording the event, by going through the script, deciding what will be the camera angles and sometimes even instructing the actors or participants before the event starts[3]. Furthermore, in the real event the television director can instruct the camera operators to change the view. However, the simulator encompasses only the topic of decision making at the time of broadcasting that is related to which camera to show to the public at a given time and henceforth we are not looking into the script manipulation nor actor guiding.

2.2.2 Intelligent Multi-camera surveillance system

The second example of a multi-camera setup is an intelligent video surveillance system. Conventional video surveillance, considers that multiple cameras are transmitting the video stream to security monitoring room where a human operator is assessing the camera feeds. In case the area of monitoring is big, then more cameras and operators will be needed for monitoring the area. The person responsible for surveillance can also sometimes manipulate the cameras by changing the camera angle they show and/or zoom in the image. An intelligent multi-camera system, on other hand, considers that the video monitoring and assessing is done by a computer. In order to make such an approach possible, the strategy is to use image processing. An example of simple image processing is to differentiate the changing pixels in image [4]. One of the methods is illustrated in figure 2.

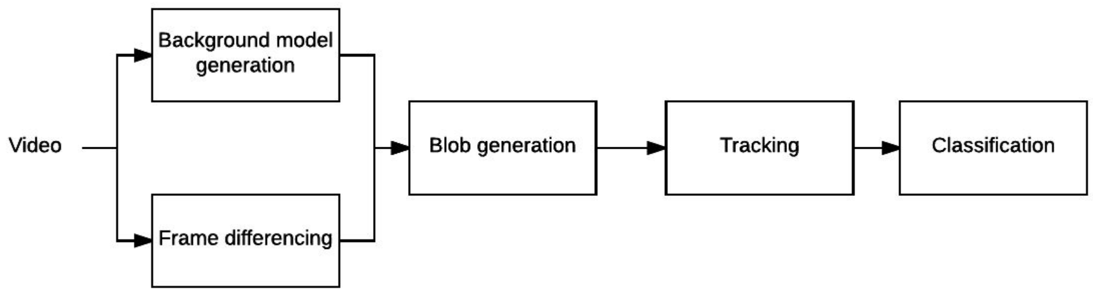


Figure 2. Video content analysis. Constructed from [4].

The first step is to differentiate the video image on foreground and background. The background is constantly updated from the received video stream, while the foreground is defined by pixels that are statistically different from the background. The foreground pixels are grouped into blobs, which are tracked and classified over the span of time. This type of processing generates high-level metadata about the video frames. One can identify if there is an object in the scene and track it. Since this type of image processing is done on the camera devices, the cameras can send the metadata to the control center instead of transmitting high quality video stream. The control center then assesses the metadata and according to user-defined rules can initiate an alert.

The camera setup requires complicated calibrations that in case of outdoor installations are based around finding similarities in picture of a map of the area and camera views that are monitoring said area. For the indoor installation the video frame coordinates are adjusted according to the blueprint of the indoor area. Furthermore, the camera calibrations are also easier to be done if the cameras share some of the view which they are transmitting. This condition is very similar to one that is in the live broadcasting scenarios of television shows.

From the television director simulator's perspective this topic was interesting to research for one of the additional features that was mentioned during the planning phase of the simulator. The idea of a feature was that there would be a certain assessment template for the choices of cameras that student makes in order to ease the assessment for the teacher and save time on grading the assignments. Teacher could insert timeframes at which time to show only certain camera and/or when a certain camera should never be shown. With image processing. the application could ask for high level rules of correct and wrong images from teacher. The image processing could then generate a similar type of high level metadata about input cameras to determine what would be the assessment template for the assignment. The background in the case of a studio show is considered to be set, and the foreground would be then determined as the participants of the show. Of course defining the rules for determining the quality of the image might require more advanced image processing which could analyze the shape, color or texture of objects.

A review of Xiaogang Wang on intelligent multi-camera video surveillance shows that this is possible, however not without limitations [5]. Errors in object tracking occur if cameras have different lighting exposures or if the area that is being monitored is too crowded.

2.2.3 Youtube and sports web-service approaches

Another interesting multi-camera streaming solution was to provide the control on what to view for the client himself. The role of the television director in such an approach is omitted and client has full control over the input videos. Youtube had this as an experimental feature “Choose your view” for US residents with desktop PCs in 2015 [8]. Apart from Youtube many sports streaming web-services (such as MLB.tv) have implemented this feature. After assessing the feature, it seemed as if this solution was perfect to make a television director simulator, since it fits the idea of practising the television director work remotely. With our own small adjustments we could change the feature to record the end result according to user inputs.

However, it occurs that there are no articles about the implementation behind the concept, or they are very difficult to find. Thus it was hard to understand how to implement such a feature. Nonetheless, user feedback and unofficial implementation response gave some insight on the technology behind it [6]. First of all, since the experiment was done by serving the recordings of the musical performance of Madilyn Bailey, many users complained that when switching the video, the audio was stuttering [9]. Furthermore, the unofficial response from Mark D [6], a filmmaker who works closely with Youtube, stated that one main video was being transmitted to the client, while others were transmitted as low-quality thumbnails for the input camera view. It is assumed that each of the videos was synchronized on the youtube server and upon switch the individual video was streamed to the client with its own audio track. The thumbnails in this setup did not have the audio and when the switch was initiated by the client, the audio stutter appeared, because the audio track needed be changed to another videos’ audio. If we were to implement the television director simulator in such a manner, the video streaming methods needed to be investigated next.

2.3 Video streaming methods

Video-streaming has several options that we needed to understand and assess. In addition to the benefits and drawbacks that arise due to the technology capabilities of the streaming methods, the methods should also be assessed by complexity. The complexity of each streaming method might affect the time required for implementing it. Therefore, we also had to take the complexity of each streaming method into account in order to finish the project during the time frame we were provided.

The situation was such that there were four video files that should be delivered to the student synchronously and two or one of them displayed another one or two times, which would be the preview and record screens. In the end, this results in six videos being displayed and they all need to be synchronised. Also the switching between the cameras should be as smooth and quick as possible. Furthermore, the PC and connection capabilities should be taken into account, as it will be students PCs and internet connections that will be using the application.

Therefore the first issue that came to mind, was that we could not just stream six video feeds to a client, since students might have limited network capabilities - it would be almost the same as having six youtube videos open at the same time. Not only could this mean possible frame drops, lags and poor video quality, but if student will have a connection that has a limit to the bandwidth usage, it would be wasted in just a couple of assignments (or even one).

For streaming there were three main methods. The methods had different implementation complexities as well as not every method was supported by every video player.

2.3.1 RTSP/RTMP streaming

True streaming is when video data goes through media servers that are specially set up for transferring audio and video data. Wowza media systems for example provides such a solution. The streaming server handles video transmission to the client and controlling the video stream based on client inputs, such as play, pause or stop. In this method the server forwards the part of the video that is currently being received by it with a small buffer, in order to allow smooth playback [16].

2.3.2 Progressive download

Progressive download means that a video file is downloaded frame by frame from the start of the video to the client machine and while being downloaded it is being played for the client [15]. This makes it possible to play the video without having to wait for the full download of the video file. However, it is impossible to skip certain parts of the videos and start playing the file from the middle, unless that part is already downloaded. Furthermore, this method requires downloading the whole video file in order to view it, which could lead to memory problems at the client system.

2.3.3 Adaptive streaming

Adaptive streaming means that the original video stream is encoded into different bitrates each with its own resolution and thus quality and data size [17]. Client application can switch between the video streams on the fly based on client preferences, which is usually network connection speed. This allows for the video stream to be delivered without frames being dropped, despite the network connection speed. However, if the network connection speed is extremely poor, the adaptive streaming will not help to accommodate that and the video might just freeze. Fortunately, since the director simulator is done in Finland for students residing in the said country, specifically for Metropolia students, we assume that they are using HOAS housing services, which provide a free internet connection of 50 Mbps. Thus if we were to use adaptive streaming, students should be able to receive at least one or two high-quality video streams.

For the television director simulator the true streaming method can save bandwidth in case student wishes to interrupt the assignment. Adaptive streaming seems to be a better option than true streaming as in addition to saving the bandwidth in the case of assignment interruption, it can also adapt to the connection speed that the client has. The extra feature which is not really needed for the director simulator is the ability to start playing the video from the middle, since television director should always decide what is shown to the public single-view output.

Even though progressive download seems to be somewhat primitive, it could prove to be more useful for the project than other methods. Since the television director simulator needs to produce single-view results form students' choices, most probably the application will have to edit it from the camera inputs. In the case of progressive download, where the video file is stored in the local system, video editing might be done on the client side in order to compensate for the server resources. However, such a scenario requires that the student has a powerful enough computer and that he or she will allow the web-application to use those resources for the video editing of his or her assignment. The drawback of progressive download where video cannot be played from the middle can be ignored as the student will need to monitor the camera outputs from the beginning.

2.4 Video players

Not every streaming method is supported by every video player. Furthermore, in order for the television director simulator to have an appropriate graphical user interface that corresponds to real life studio, the video player needs to have capability to provide access for the video frame data. Moreover, for the potential extra features, such as an intelligent assessment template, which would recognize good and bad camera frames, any form of image processing provided by the player is considered to be a plus. For the video players there were several options, but we pinpointed three of them as the most prominent.

2.4.1 Video Lan Client media player

Video Lan Client media player (or VLC for short) is free video player that the developer can plug in into his web-application. In order for the client to use the player he would need to download and install the VLC plug-in on his machine. For the web application it seems as extra work to download extra software and in case the user would want to access the television director simulator on his mobile phone or tablet, he would need to have the VLC plug-in installed also on those devices. An additional downside in terms of video management was that we could not access the frame data through the player in order to synchronize the video streams, which would mean more work on the server side. The benefit of the player is that it comes with good built-in controls and a possibility of live video streaming from the client machine. Therefore, for the live streaming feature VLC seemed a good option as we could potentially use the same application to set up camera transmissions from the studio.

Furthermore, the software can also be used in an offline application, meaning that if we were to do a desktop application the VLC player was a potential candidate for our application. The student could install the player when installing our application.

Additional tests were done with this player in the web format in order to figure out how the delay between the video streams would look like if we launched two players at the same time. The delay was definitely present and what was interesting, it was not systematic. The delay between first and second video went from being behind to being ahead randomly. Therefore, additional means for synchronization were definitely required.

2.4.2 Adobe Flash Player

One of the leaders on the market is the Adobe flash player that includes advanced features for streaming, frame manipulation and good customer support. Even though this player required plugin installation, it did not seem to be a strong drawback, since anyone using youtube has to have this player installed. The major drawback was its price. At the time, implementing the player was not free, but required a monthly fee and we had a synchronization problem to solve. It might be that solving the synchronization problem and graphical user interface would not require Adobe player and thus we were sceptical in acquiring it.

2.4.3 HTML5 video player

HTML has an integrated video player. The file path to the video has to be passed into HTML tag in order for the video to be displayed on the web-page - very simple. Obviously this approach of a displaying video file on the web page is free. The player does not require any plugins to be installed on the computer, therefore the application could also run on brand new machine. To our surprise we found out that the video frame data of the player can be accessed directly through JavaScript in a very simple manner. Because of this, developers can decide to hide the player from the view and display the video-frames on an HTML canvas, however they want, with custom video player controls.
Drawback in the case of HTML (at least at the time of development) is that it uses progressive streaming method. Since the file is downloaded to a user machine, it means the user computer should have enough temporary space to use the application. In the case of desktop PCs and laptops this does not seem to be big a issue. However if the student decides to use the application on mobile phone or tablet, the quality of the video might have to be very low in this case. On the other hand, mobile devices are small and therefore choosing low quality video in that case is beneficial.

2.5 Multi-camera director simulator

In a multi-camera director simulator students will have to work with a ready-made setup, to experience the director’s work. As already mentioned the students could potentially do home assignments independently and at any time they want. The setup in this case is defined by four pre-recorded camera inputs. The camera inputs are produced at Metropolia University of Applied Sciences in the multi-media studio. Since the setup at the time had only four cameras and all the test assignment materials that we received from Pekka Korvenoja consisted of only four cameras, we decided to limit the application camera input number accordingly. Furthermore, the video recordings had to be shared with students somehow. From the research we understood that there were two different possibilities to implement the television director simulator: web-application and offline desktop application.

In the case of an offline desktop application the students would have to do the assignment either at school where the teacher could share the assignment material (camera recordings) easily, or if the students were to do the assignment at home, they would need to download the materials first. The offline desktop application had its benefits and drawbacks. The easiness of making the desktop application came from the fact that we would not have to account for the bandwidth problems and connection speeds. It is assumed that downloading the materials from the school server would not require as reliable and fast of a connection as it would for streaming the video. Furthermore, quite a high video quality can be achieved with such setup.

However, in order to perform video editing tasks a quite powerful computer is needed. If student wanted to do the assignment at home, he would have to invest in his processor, RAM and hard drive. The video editing is a resource-heavy task and the camera inputs of high quality as well as the result needs storage space. In a case where the student does not have enough computer resources, the video editing that application will perform might take too much time or it might just freeze the whole system. Although the video editing could be sped up at the cost of the video quality produced, we thought that student had to do too much extra-work irrelevant to television directors’ duties in order to practice the directing tasks. Furthermore, when student has produced the end result, he would need to share it with teacher.

Therefore, student would need to upload the assignment to the school server and depending on the size of the result and connection speed, this might take a considerable amount of time.

The second option was to make a web application. With a web application student would have to have a reliable and fast enough network connection. As mentioned for the Metropolia students who use HOAS services the network connection speed is 50 Mbps, which should be enough in order to accommodate one or two good quality video streams. Students who do not use the HOAS provided network connection, will have to either do the assignment at school when they can or upgrade their network connection. Accommodating the application for network speed and bandwidth is in itself a challenge.

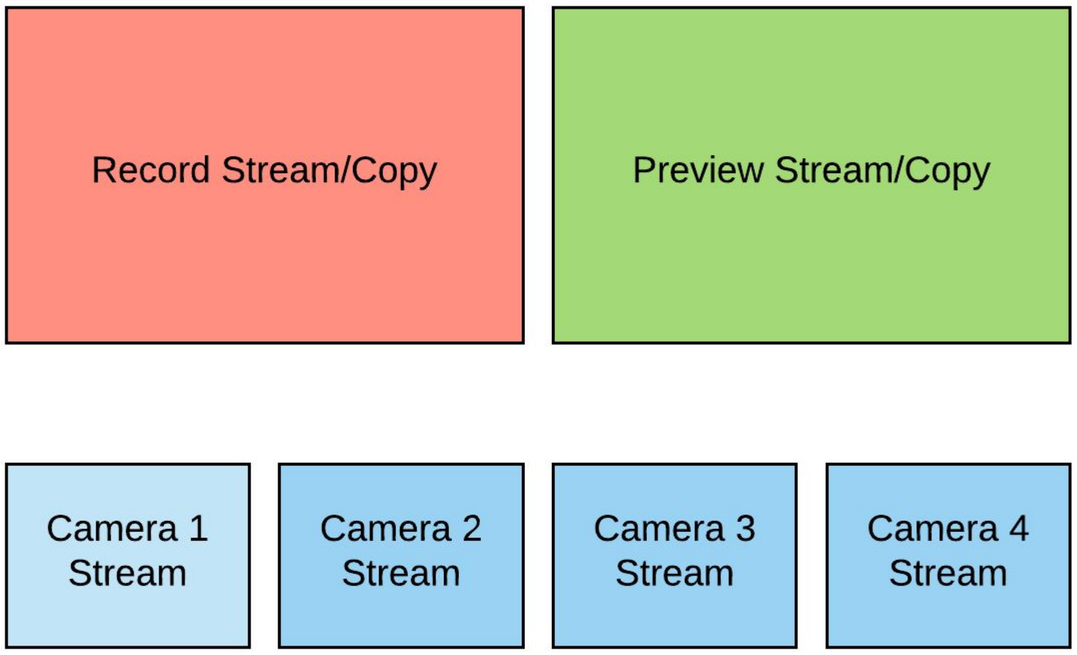


Figure 3. Demonstration of simple graphical user interface with technologies behind them.

For example, as can be seen from figure 3, we have four video inputs which need to be streamed to the client, as well as preview screen and record screen, which are to display some of the inputs additional time. The preview and record screens are either implemented as additional streams or copies of one of the original four input streams. Therefore, the web-applications major drawback was that it was more complex to implement than offline desktop application. However, a web-based approach, if done correctly, has various advantages over the offline desktop application, from the user experience perspective. First of all, student would not have to download the video recordings, at least not directly as in case of progressive streaming the files are still downloaded to the client computer. Nonetheless, the application should have an interface for the teacher to upload the assignments. Furthermore, students will not have to invest in a powerful computer in order to use the application, as all the video editing is considered to be done on the server side. On the other hand this also means that the server computer needs to be powerful enough if it is to perform several video editing tasks. Finally, student will not have to worry about uploading the assignment, as the result will be automatically available in the web-application when student finishes his assignment.

Despite which approach we choose, the application will need to synchronize the videos. When assessing the VLC video player, we tried to launch two videos at the same time and results showed that this type of approach will not work, as the delays between videos were occurring randomly and the magnitude was inconsistent. In a case where the videos are not synchronized and the same camera input were to play in record and preview screens, the user might notice a slight delay between the two. Furthermore, if we would use the youtube approach of multi-camera view selecting, the input cameras will probably have each their own audio track and the switching itself has to be flawless in order not to have any audio stutters. Therefore, even if we would use only one audio track to tackle the problem of audio stuttering, it would mean that one audio track needs to be synchronized with other videos. Otherwise, student might also notice that speech is out of sync with certain camera inputs. Such an application property can determine if the television director student makes a choice whether to show the camera to the public or not. In real-life broadcast events the synchronization problems might be caused by the camera equipment for example. Therefore, in a case where the input camera which is out of sync is supposed to be the correct one for the student to choose at a given moment, the synchronization problems caused by the simulator might lead him or her into a wrong conclusion to choose a different camera. Thus, television director simulator needs to have synchronized video streams.

Furthermore, the results of student work should be saved in order for the teacher to have a possibility to provide feedback for the students on their work, and give demonstrations for future students of good and poor examples of television directing. In both cases the Metropolia server has to somehow store these numerous examples. Therefore, director simulator should have certain rules for controlling the result storing, Metropolia should expand storage space required for the results or we would have to find other solutions.

3 Director simulator application components

After assessing the benefits and drawbacks of both the web-based approach and offline desktop application, we came to the conclusion that we had to do a web-application. The drawbacks of the offline desktop applications were all client-experience centered. The whole purpose of the television director simulator is to make it easier for the student to experience the television directors' work. With the desktop application downloading everything, installing the application, investing money for the computer system and then uploading the end result is very exhausting and negative user-experience. Even though the web-based approach was harder, we believed that we could manage to produce a working product within the provided time limit of six months.

For the web-application there are many different technologies to choose from. The choice over technologies also depends on what the purpose and demands of the application are. Furthermore, the complexity of technologies in respect to their capabilities needed to be evaluated also. As developers we also needed to account for our past experience. In terms of web-application I had mainly used Node.js with Express and MongoDB. For the frontend development I was using React. All these technologies can be used with JavaScript. My partner Oleg Batalin was mainly a Java developer. There were three main technologies that were dominating the market at the time:

- ASP.NET.
- LAMP-stack.
- MEAN-stack.

ASP.NET is Microsoft supported framework for developing web-applications. ASP.NET uses C#, with which not me nor my partner had any extensive experience with. Furthermore, ASP.NET was designed to use thread-based request processing. When multiple users issue several requests from the client application to the server, the server has several options for handling them. The thread-based method requires from a developer to allocate a separate thread for each of the requests from the available pool of threads [10].

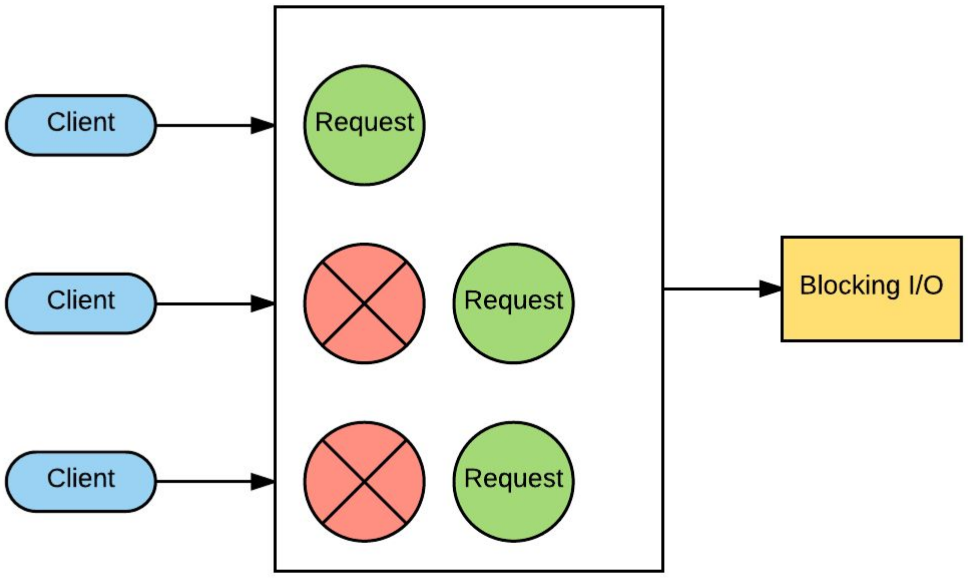


Figure 4. Example of thread-based client request management.

In addition, since the pool size of threads is limited, the threads may be allocated for each of the clients instead of each of the requests or each of the interfaces. In any case, there is a danger that client might issue a new request, before the previous one finishes executing. As illustrated in figure 4, with exhausted resources client will have to wait until the old request has been finished executing, before his new request is handled. In a television director simulator the student will need to be able to switch cameras without restrictions. Furthermore, the assignment completion should not be restricted to a few students at a time. If we were to allocate multiple threads to a single student, we would run out of resources pretty fast and would not be able to serve other students. There was a threat that such an approach would result in a slow application speed, queue time and overall poor user-experience. Moreover, thread based method requires to have costly server, which can support multi-thread applications. Solutions for accommodating the new users, consider investing more in the server resources, which would also mean that either Metropolia will invest enough in the server side of application or it will have poor service. Finally, the thread-based method requires that a developer has to remember to map the threads to requests and thus it adds more complexity to the development of the application.

LAMP is an abbreviation of Linux, Apache, MySQL and “PHP, Perl and Python”. The stack is considered to be old, but stable. MySQL can be useful database depending on application needs and Python can provide useful tools for advanced image processing. However, Apache is a server technology that also utilizes the thread based method, but in a slightly different manner [11]. Apache spawns a process with its own thread pool in case the process runs out of threads. However, this type of approach still requires the investment into a processor and memory of the server in order to accommodate new users and we were expecting the same complexity associated with the thread manipulations.

Both of these technologies are old, complex to use, difficult to learn and require expensive approaches for serving many users at the same time. The MEAN-stack on the other hand was quite unique in these aspects. First of all, it was relatively new on the market and thus we had more interest in using it. MEAN is an abbreviation of MongoDB, Express, Angularjs and Node.js. All of these technologies can be utilized with JavaScript as a programming language. From language perspective MEAN-stack had strong advantage over LAMP and ASP.NET. Even though ASP.NET uses C# for the server side, the front end of the application still needs to be done in JavaScript, meaning that we had to use multiple languages anyway if we were to use ASP.NET. Therefore, MEAN-stack is also easier in learning than the other two contenders and because my partner was mainly a Java developer, the learning curve for the MEAN-stack would be more gradual for him than compared to the other two. Finally, Node.js uses a different approach for request management, which allows great scalability in case of user growth of the application and developer does not have to worry about thread manipulation, which adds to the simplicity of the technology.

3.1 Node.js

For managing the requests Node.js is using event-driven asynchronous I/O model. The Node.js still utilizes multiple threads, but in a smarter and simpler manner. The Node.js code that developer writes is running on a single thread and the methods for processing requests are asynchronous functions with callbacks. These methods are considered to be events, because when the request arrives at the server, a certain method is called, or in other words the event is fired.

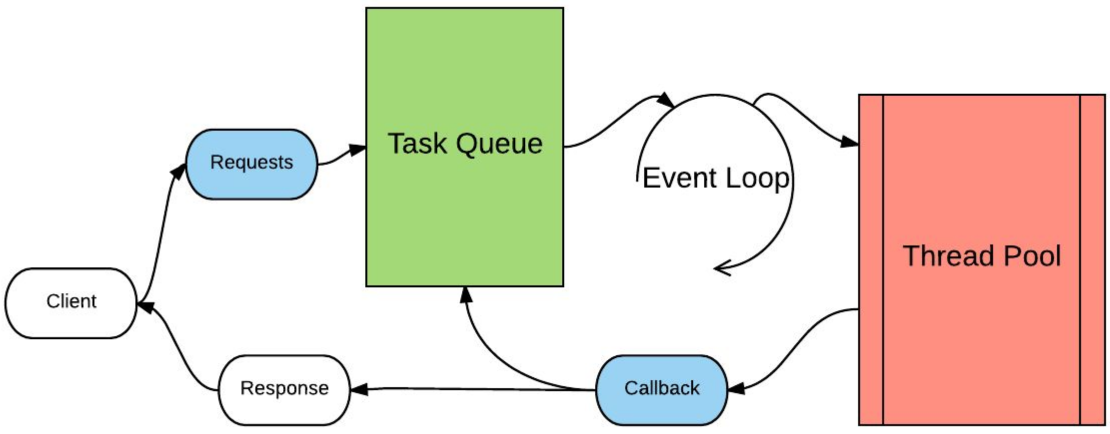


Figure 5. Event-driven asynchronous model.

As can be seen from figure 5, when the event is fired the execution of that method is put on hold in a task queue, since Node.js runs on a single thread. The main thread (where Node.js is operating) runs an event loop, which is hidden from the developer. The event loop picks the event from the top of the queue and allocates a thread from a thread pool for execution of the event handler, which was defined by the developer (processing of the client request). Once the event finishes the execution in the thread, it fires the callback function, while the thread is made available for later use. The callback function either returns a response to the client or fires another event, which will be put at the end of the task queue. Such an approach means that one request does not delay other requests, while requests that are executed fast enough compensate for the slower ones, allowing to effectively use the computer resources and serve more users than with the conventional thread-based approach. The event-driven asynchronous model does not only separate clients or requests on different threads but also the request processing steps. For example, in television director simulator student will want to view all the available assignments. Student issues a get request to the backend of the application, which is operating on Node.js framework. The processing steps for fetching the information from database are as follows:

1. Get data request.
2. Connect to database.
3. Fetch data from database.
4. Return data.

The first step is the first event that is fired and put in the event queue. After execution of that event it will return a callback function, which is the second event in that queue. That callback is then placed in the event queue. The process repeats until the last step is executed where the data is sent to the client. This means that processing steps are executed on separate threads, and each function and callback will first have to go through the event queue and be picked up by the event handler in order for it to be processed. It might seem at first that this type of an approach slows down the execution of the request. However, the event loop has a high execution frequency and it will allocate the events to threads quite fast. The main requirement for the developer is not to write code in the main thread that would slow down its execution and use asynchronous function implementations with callbacks. The simplicity of the Node.js comes from the fact that the thread allocation is handled by Node.js automatically, through accessing C++ apis.

Node.js itself has a relatively low-level abstraction, meaning that the system itself does not have many pre-built features for network request management. It is a framework and one can use basic tools to make his own network request managers. Therefore, Node.js is not usually used without the help of additional libraries, because otherwise there would be too much work to do for developers. Node.js libraries are called node modules and they are redistributed through node package management ecosystem abbreviated as npm. The open source node modules do not cost anything. However the open source license that the module has might state that an application, which uses the said library, cannot generate revenue or if it is to used commercially. Then the module might have to be bought. The size of the node modules is usually small and thus including the node modules in your development environment is quite easy and fast. Some modules might have additional dependencies on them and it might slow down the development process a bit, as the developer needs to check if he has the dependencies with appropriate versions installed. Furthermore, these types of modules are on an average bigger in size. However, the bulky modules, carry various functionality with them, which is supposed to speed up the overall application development process. Once the application is ready and node modules are in check, the deployment to the server is relatively easy, since the npm can fetch, with a single command, all the dependencies that are listed in package.json, a manifest file located at the root of the project directory.

3.2 Express

One of the most frequent node modules that is used is Express. Express has many helpful features for building mobile and web applications. It is constantly updated and it has a higher level of abstraction compared to raw Nodejs when it comes to developing network applications.

Many other popular frameworks are built on top of Express to even further make the development-specific network applications easier. LoopBack is one example for building REST apis, which I had experienced myself previously.

3.3 MongoDB

MongoDB is a non-relational database system. Contrary to the MySQL it supports dynamic models (one can add fields to the documents or delete fields from documents at any point of database management). The elements of each database are illustrated in table 1, in correspondence to their functionality.

Table.1. Corresponding element names in MongoDB in MySQL. [14]

MySQL	MongoDB
Table	Collection
Row	Document
Column	Field

A major difference between MySQL and MongoDB is that the latter stores its entries in BSON format, (JSON in a binary encoded format) while MySQL stores the data in tables. The difference is that each entry, or documents as they are properly called, contains information about its own data. An example can be seen in listing 1.

```
{
      "name" : "Michael",
      "age" : "60",
      "weight" : "60",
      "height" : "180"
}
```

Listing.1. MongoDB document example in JSON format.

Such a document-oriented storing method has two effects on the database system. The benefit is that the data can be automatically split between different storage systems, which would allow easy scalability. The negative side is low data integrity. If someone was able to retrieve document entry from MongoDB, he would know exactly which data belongs to which field, while if someone retrieved row entry from MySQL database, he would not know to which column the data belongs to. The truth is many applications require more scalability than data integrity nowadays (and ours is such).

Another drawback of MongoDB is that it has no good binding between different data models. One can embed a document in another document, but this would mean that entries can get extremely large if we were to bind documents in this way. The design principle for MongoDB is that models are defined outside the database in the application logic and if entries do get too large the data model structure needs to be re-designed. It could be better for example to have a field in a document, which contains id of the other document in another collection.

3.4 Angular 1.0

AngularJS is an open-source JavaScript framework for developing single page web-applications. Therefore it allows such frontend view manipulation that client does not have to wait for the content to be loaded in a new instance or page. This allows to build dynamic and interactive graphical user interfaces. If we are to make a web-based television director simulator, such a framework would be ideal to make the user interface for assignments that could imitate the real-life production studio.

Setting up an Angular application requires three main steps. The first one is attaching an application to a static web-page which will be manipulated. After that the framework needs to know which section (HTML element body or div usually) will contain all the switching views. After ng-view is allocated, all the developer now needs to do is create the views. The views have always an Angular controller written in JavaScript through which the developer manipulates the actual view. The developer can also have a static HTML content that he can manipulate in order to avoid creating everything in the controller itself.

3.5 FFmpeg

FFmpeg is a software for video manipulation and editing. FFmpeg requires different codecs depending on the task that it is going to be used for. Even though the setup can be troublesome, the software is quite comfortable and easy to use. One can manipulate videos however he wants with such a tool: cut videos, combine them into one, convert them, append them one after another, manage audio tracks by removing, shifting, combining and/or adding multiple audio tracks. FFmpeg supports various formats such as mpg, mp4, avi and mov.

For the television director simulator we would need to manage the videos at least to some extent. If we were to stream the input cameras to the student and then, based on his inputs, produce a result on the server side, the FFmpeg would be a perfect tool for such a task.

3.6 Nginx

As well as Node.js, nginx is event-driven asynchronous server runtime, which makes it good option for handling multiple users at the same time. The nginx is known for its stability and in this project it is used as a proxy server to route the requests from the client to the application and back. Nginx has low resource consumption and therefore adding it as an extra security layer to the project will not do harm. If the application crashes nginx will still be operational and can inform the client about “maintenance” or “technical difficulties”.

4 Implementation

There were many challenges in building a simulator, which would be as close as possible to the “real thing” and provide students with a possibility to train independently without pre-scheduling the assignments.

- Graphical user interface.
- Video to video synchronization.
- Audio to video synchronization.
- Assignment management.
- Result production and storing.
- File upload.

The approaches that we used in the final product are quite innovative in terms of implementation purpose, meaning that they were not used in a multi-camera streaming environment before (at least at the time). Solutions for complicated problems include the explanation of what we tried and what we ended up with.

4.1 Simulator user interface design

One of the first tasks was to understand how the director simulator should look like. There are many multi-camera directing studios around the world with varying number of input cameras. Several designs had the input cameras on the left, right and/or the bottom of the recording screen. Some designs did not have the preview screen.

Since students would have access to the Metropolia studio on some occasions, the design was decided to be as it is in Metropolia: 4 video inputs in a row below two main screens, one preview window on the right side (green on figure 6) which indicates to which camera should the next switch happen and one record window on the left side (red on figure 6), which shows what the public will see as the end result.

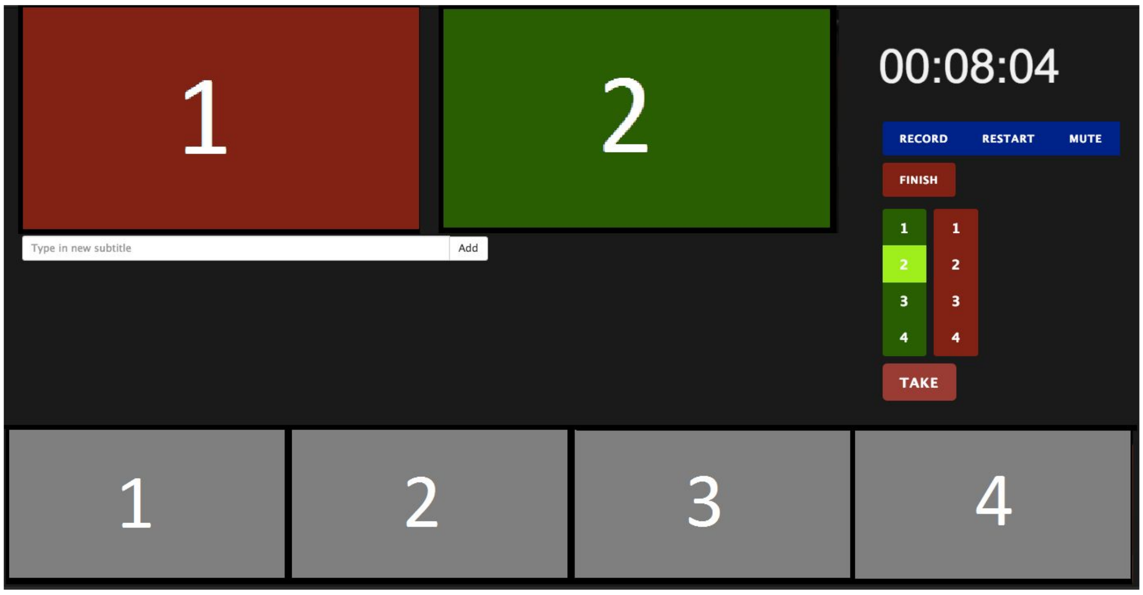


Figure 6. Graphical user-interface for the director-simulator, first prototype.

As for controls over the video streams - four buttons for each of the window: red and green for record and preview respectively and “Take” button to change the record input stream to that which the preview is showing. In addition, the user interface has to have the duration of the stream. In a real streaming environment the time limit acts more as a guideline for the director, though a strict one as a good director always tries to fit into the time limit. The timer counts downwards, beginning from the length of the video recordings. Therefore, it is always good for the student to see how much time there is left until the assignment ends. An important point to note here also is that all recordings are of the same length and therefore the timer accommodates all of them.

4.2 Video Synchronization

After assessing the video streaming methods and video players, the HTML video player with progressive streaming had several advantages over the others: no plugins, free, has the ability to provide the frame data for possible synchronization and custom video layout, and finally custom controls. This also gave us a big insight into solving the synchronization and bandwidth problems.

A great feature that an HTML player has is that we can hide the player and display the video frame contents on the canvas, using simple JavaScript. This means that when student wants to switch the preview a camera view to the record screen, in other words press the take button, or if he wants to change what the preview window is showing, we could replicate the video frame data from one of the four input windows to the preview and/or the record windows. This would reduce the required amount of video streams by two from six (four video inputs, preview plus record) and as a result the required bandwidth for good quality video streams, since the video on the preview and record screens is just a graphical repetition of one of the inputs.

The next issue to solve was the synchronization problem. As already discussed, the main purpose of the application was to serve the recordings of the “play” sessions. This means that the server has access to the video files before they are streamed to the student, as they first have to be loaded to the system. In addition, a client HTML player with the help of canvas can also display only a part of the video stream frame. Therefore, if there is a need, only a quarter of the whole video could be displayed. Furthermore, we figured out that we could use FFmpeg to combine four video recordings into one synchronized video recording on the server side and then that video recording could be streamed to the client side and “divided” there into screens with the help of HTML player and canvas.

It might seem at first that this method does not improve the bandwidth issues, as the resulting video should be as big as four combined videos, from the data perspective at least. However, progressive streaming which HTML is using is operating over the tcp protocol. The way TCP protocol manages the data-packets is generating way more traffic for four different data streams, than if those four data streams would be combined and here is why. Apart from data, the tcp packet contains 20-60 bytes of header information, with four video streams this can add up to 80-240 bytes of header information. Even though this is a low amount for the current network bandwidth, with a constant video stream this can add up to a substantial amount and can be the deciding factor of the service quality in case the client is using a limited bandwidth connection.

In addition, the tcp data link establishment requires “multi-step handshake process”[12]. This means that both endpoints of the link need to exchange several packets before beginning the transmission, and with four different data links, this is four times more packets to be exchanged; thus it can delay the assignment start time.

Furthermore, in TCP the sender of the packets will resend the data-packet if it did not reach the destination on the contrary to UDP protocol, where sender does not care if receiver got the packet or not. If synchronization would be done on the client side, with four video streams, there is a high probability of causing delays. In the networking the acceptable packet loss is considered to be less than two percent. Good network connection has less than one percent [13]. Now considering that student has a relatively good connection quality, potentially each of the video streams has probability of one percent to lose a packet. In case of progressive streaming if there is not enough data already downloaded for viewing (big enough buffer), the video will stutter as a result of missing data packets. If there are four streams with error probability of one percent that error probability rises to 1-0.99^4 = 0.0394, which is an around four percent error rate instead of one. Normally if a network yields a packet loss of near five percent or more, then the issue needs to be investigated for it is considered a bad quality of network connection [13]. This also means that with such synchronization on the client side the buffer for “pre-downloaded” content has to be substantially higher, in order to account for the error mitigation. The packet loss depends on many things, while less than one percent is a relatively acceptable margin, and anything higher would have produced poor user experience.

4.3 Audio synchronization

For the audio synchronization solving the problem was relatively easy. The Metropolia studio stored the original input videos in such a way that only first camera recording had the audio track of the whole session. Therefore, we could take the audio track from the first camera recording and add it to the combined assignment video file. An important case to note here is that if there is a need for producing the final result by cutting and combining the original input videos at certain timestamps, then in addition the audio track will need to be copied from the first video over the whole end result. In case the copying is omitted, the end result will be muted at the points where all the other cameras are to be presented, except the first one.

4.4 Assignment management

Since now we have one combined video file on the server side, we can give the location of that file to the HTML player and hide the player. With the canvas we can now define where we want to render the combined video frames. The frames are copied three times for each of the HTML canvases: one for the lower panel (four input videos), one for the preview screen and another for the record screen. The preview and record screen show only part of the frame, because the other part is out-of-bounds of canvas resolution. The lower panel of four input videos is actually the video itself that is located on the server. We hide the original player in order to disable the controls for the video and add our custom labels. In addition to the labels, other effects can also be added using HTML canvas. This is also how the custom subtitles are added to the video.

When the student switches the camera, we shift the frame on the preview/record screen accordingly. The frame that is displayed is the same, but now that the different part of it is shown it leaves a feeling that the camera was switched. The student input is recorded whenever they press take button. The camera and timestamp are then sent to the backend for further processing. The same happens with subtitles. Student enters a custom string which they can display for a fixed amount of time (3,5,7 seconds). The string, option and timestamp are then sent to the backend.

4.5 Result management

The application had to face a large amount of student "recordings", meaning each student had to do the assignment and potentially have the end result saved. The first method, which came to mind was to process the student input and cut the videos from original files according to that input, while the student is doing the assignment. Editing the end result while student is doing the assignment would allow almost immediate view of that result. However, cutting videos together is a resource-heavy task on the computer system, from processor and RAM perspective. This means that if there are multiple students doing each of his own assignment, the resource requirement will just multiply and we would have to have a very powerful server. The second method was to save the user input to the database and then later use that input to produce the end result. This would allow to queue the end result production on the backend, but the result itself will not be viewable on demand.

A common problem with both of these approaches is that despite when the end result is produced, during the assignment completion or after, it is essentially a video file, which requires a storage space. This means that either spend more money on the storage space at the server side or create rules for teacher as for when the end results would be deleted. For example, in the end of the course the teacher can delete the files produced by students. Such an approach removes the option for demonstrating past works for new students and in case there are good or very bad results, then it needs to be downloaded and saved separately in order to view it in the future.

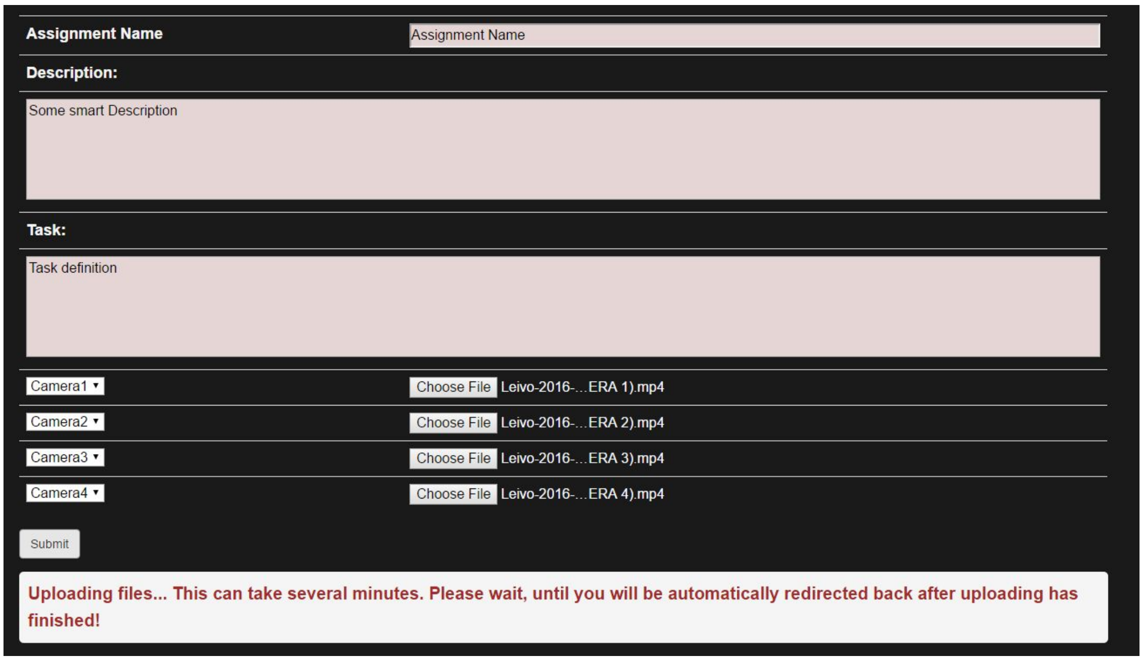
The method that was used in the end solves these problems. The end result is never produced as a video file, but rather all the student input is saved to database. Teacher can view the results using the application and upon viewing result of specific student for specific assignment, the server sends the same combined video file of four recordings for that assignment and client then uses the student input from database to "swap" the frames according to the changes that the student made during the assignments. The method of presenting the end result here is the same as showing an input video on the record screen on the assignment page. For the client the HTML and canvas combination is light from a resource perspective. Now there is no downtime for the student to view his own result, and the storage space on the server side needs to accommodate only the combined video files, which equals to the number of assignments rather than number of results. With each student completion database gets filled only with a map of the input camera number and the time when it is supposed to be shown. If there is a need for growth the MongoDB can be easily expanded with more storage space as it supports data sharding, meaning the data can be separated into chunks that do not have to be located in the same database. In the end we were amazed by the fact that we could come up with such a solution.

4.6 File upload

For the fileupload I first tried two different options. One was AngularJS own readymade file upload solution. It was easy to implement. However for uploading multiple files at once the user interface was lacking dynamic. The problem was that when the teacher presses "browse" button, he or she should choose four video files in that same window and have them uploaded through such interface. However, this is not very intuitive and I was afraid that old teachers might have even less success with such an interface.

This brought me to the second option - socket.io. Frankly speaking I was lacking in technical skills and could not implement a proper file upload with the sockets. I spent some time trying to implement the sockets, but the deadline for the file-upload feature was coming close and I did not understand what I was doing wrong. Therefore, I used a conventional HTML form for uploading files. A good thing about the HTML form is that it is easy to implement and customize. I could make the interface for uploading as intuitive as possible. The drawback of the HTML form was that I could not get the upload progress status (at least at the time). This was fine as I could display a message for the end user once the upload started that one should not leave the page. Once the upload finishes, the application will redirect the user (admin or teacher) to the assignment page. Another drawback of the HTML form was that if the server does not respond back on time, the browser will try to send the form again. Therefore, if the upload takes too much time, the client will be stuck there forever. I found a workaround for this problem by disabling the form after the upload had started, so that when the browser tried to resend the form it could not do it, since the form was inactive now.

The user-interface of the upload form allows teacher to define the assignment title, description and task. Teacher can map the four recordings to custom positions from camera one to four as can be seen from figure 7.


Assignment Name	Assignment Name
Description:	
Some smart Description	
Task:	
Task definition	
Camera1 ▾	Choose File Leivo-2016-...ERA 1).mp4
Camera2 ▾	Choose File Leivo-2016-...ERA 2).mp4
Camera3 ▾	Choose File Leivo-2016-...ERA 3).mp4
Camera4 ▾	Choose File Leivo-2016-...ERA 4).mp4

Submit

Uploading files... This can take several minutes. Please wait, until you will be automatically redirected back after uploading has finished!

Figure 7. Upload form.

Once teacher submits the assignment, he will be prompted with a message (in the bottom of figure 7), to wait until the upload has finished.

For the backend, once it receives the video files it will start editing them into one combined assignment video. The video editing on the backend is complicated process and consists of several steps.

- Converting into mpeg-2 format and renaming the files.
- Combining the mpeg-2 camera recordings into one line-view video.
- Converting the mpeg-2 to mp4.

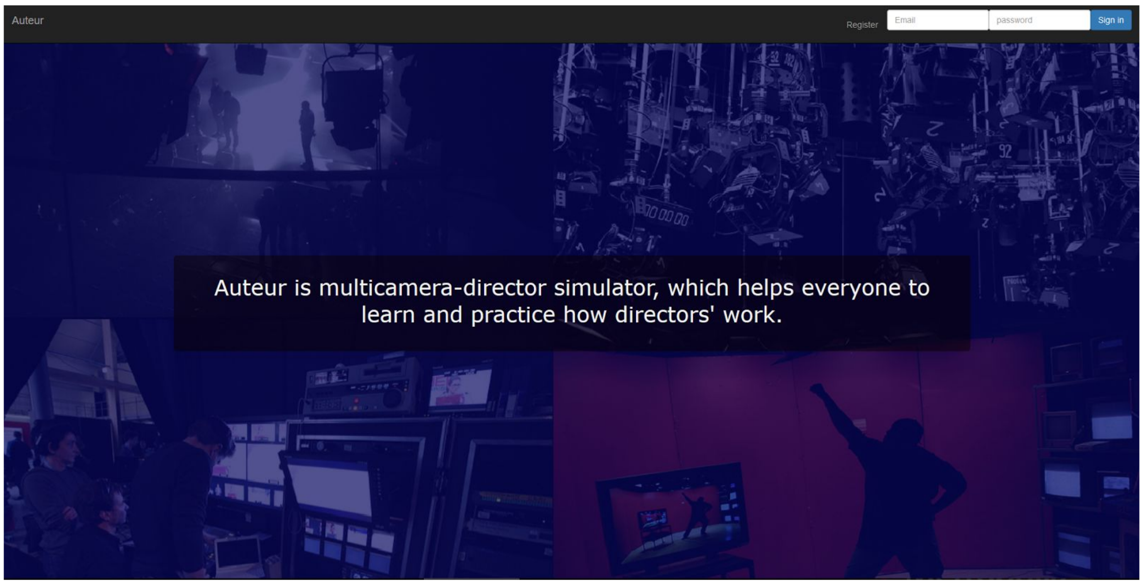
The first thing that happens is that the server converts the files from whatever format it received them into mpg (mpeg-2) and renames the files accordingly. The mpg format is in order to lower the time required for cutting and combining the videos. If the user uploads the files in .avi or mp4 (mpeg-4) format, the video formatting will take substantially more amount of time and therefore we first need to convert those user formats into mpg. Since the application also provides an option for students to choose the quality of video streams, the backend produces one assignment file for each of those qualities. Therefore, the following tasks are run for each of the qualities and the first one is combining the mpeg-2 camera recordings into one video. Currently there are only two options: 360 and 480 pixels. Once the videos have been cut, the next step is to convert them into an mp4 format, so that students could use browsers they want. The mpeg-4 format was chosen, since it is supported across all modern browsers.

After the combined video is produced, all the unnecessary files are deleted. The combined video files are kept on the server in addition to the four original converted and renamed files. The original files are kept on the server for potential download feature. In case teacher or student wants to download his or her end result, it would be cut and combined from the original video files according to the timestamps in the database. However, we did not implement this feature, since it can have a serious impact on the server stability. In case more than one person wants to download their result, there is a risk that the server would just hang due to the pressure on its computing resources. The file editing steps on the other hand are implemented using asynchronous functions with callbacks, callbacks being the implementation of consecutive steps. With this approach the video editing steps use different threads one after another.

The file upload is separate from video editing, meaning that teacher does not have to wait until the assignment video is ready in order to continue using the application. The only thing that user should wait for is the file upload to finish. Once the assignment video is prepared, the assignment will automatically appear on the assignment page with all the details.

4.7 Overall user interface and usability of the application

In order to use the application, currently anyone can register and start doing the assignments that are uploaded to the application. The application does not verify the e-mail, so the e-mail can be anything. The registration can be accessed from the landing page of the application (figure 8).

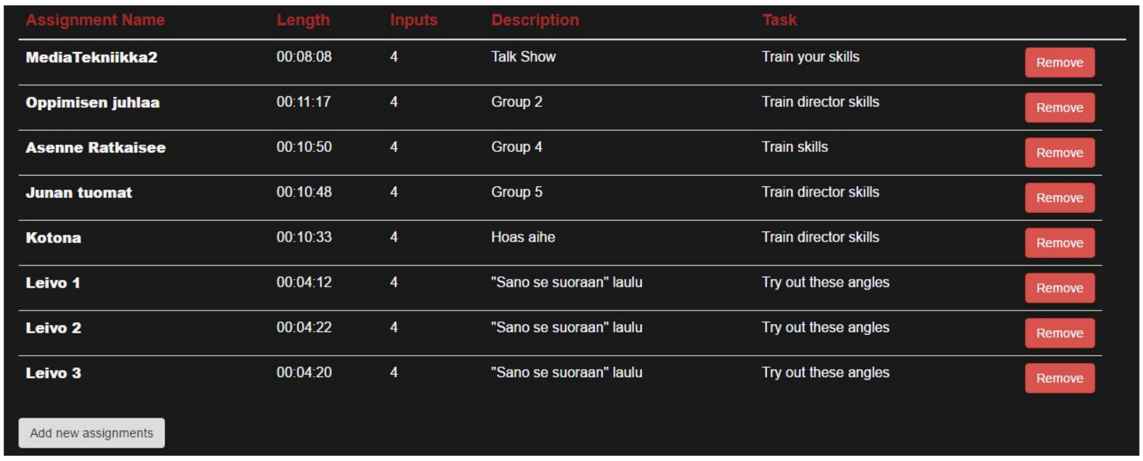


Auteur is multicamera-director simulator, which helps everyone to learn and practice how directors' work.

Figure 8. Application Landing Page.

The registered users will not be able to view other users' results nor upload new assignments. Only the admin user can perform these tasks, and whenever the teacher is registered, he or she should apply for the admin rights.

From the landing page students can also go to the assignments page to view a list of all assignments. The assignment listing contains the title, duration, description, task and number of input cameras (figure 9). One of the ideas for improvement would be to have a dynamic number of cameras available for assignment creation. Currently the number is static and has to be exactly four. Teachers can also delete the assignments, apart from creating one.

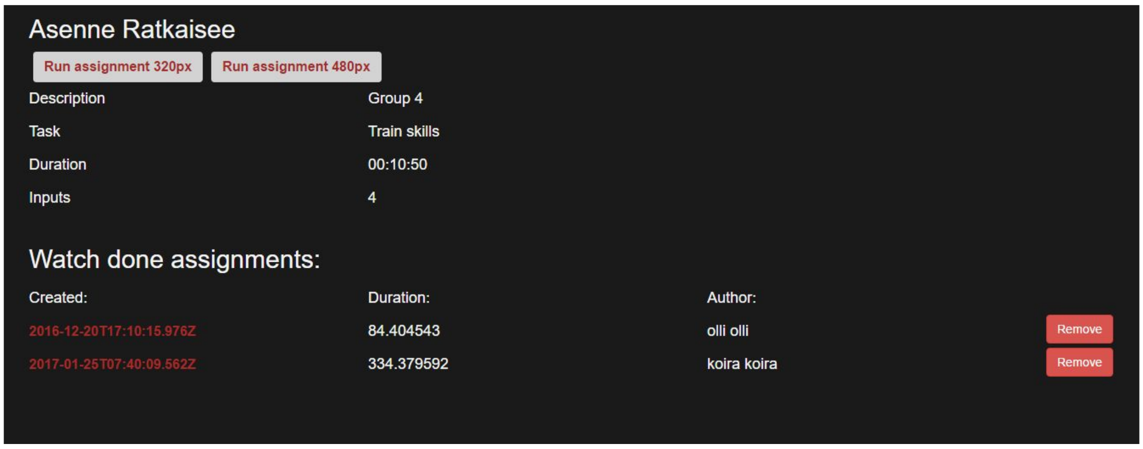


Assignment Name	Length	Inputs	Description	Task	
MediaTekniikka2	00:08:08	4	Talk Show	Train your skills	Remove
Oppimisen juhlaa	00:11:17	4	Group 2	Train director skills	Remove
Asenne Ratkaisee	00:10:50	4	Group 4	Train skills	Remove
Junan tuomat	00:10:48	4	Group 5	Train director skills	Remove
Kotona	00:10:33	4	Hoas aihe	Train director skills	Remove
Leivo 1	00:04:12	4	"Sano se suoraan" laulu	Try out these angles	Remove
Leivo 2	00:04:22	4	"Sano se suoraan" laulu	Try out these angles	Remove
Leivo 3	00:04:20	4	"Sano se suoraan" laulu	Try out these angles	Remove

Add new assignments

Figure 9. Assignment list (Admin view).

When student selects the assignment he can see the same details for the assignment, choose to run the assignment in one of the quality formats and view his previous completions on this assignment (figure 10). If student wishes, he can also delete his own results.



Asenne Ratkaisee

Run assignment 320px Run assignment 480px

Description	Group 4
Task	Train skills
Duration	00:10:50
Inputs	4

Watch done assignments:

Created:	Duration:	Author:	
2016-12-20T17:10:15.976Z	84.404543	olli olli	Remove
2017-01-25T07:40:09.562Z	334.379592	koira koira	Remove

Figure 10. Assignment details (Admin view).

Teacher can view his own completions, as well as all other users' completions, including other teachers. Teachers also have right to delete the results of their choice. Unfortunately, only highest quality available at the time is 480 pixels and quality of 720 pixels has not been tested. In case additional qualities are added to the application, the assignment creation will take more time. Once student selects an assignment quality he will be taken to the actual simulator (figure 11).

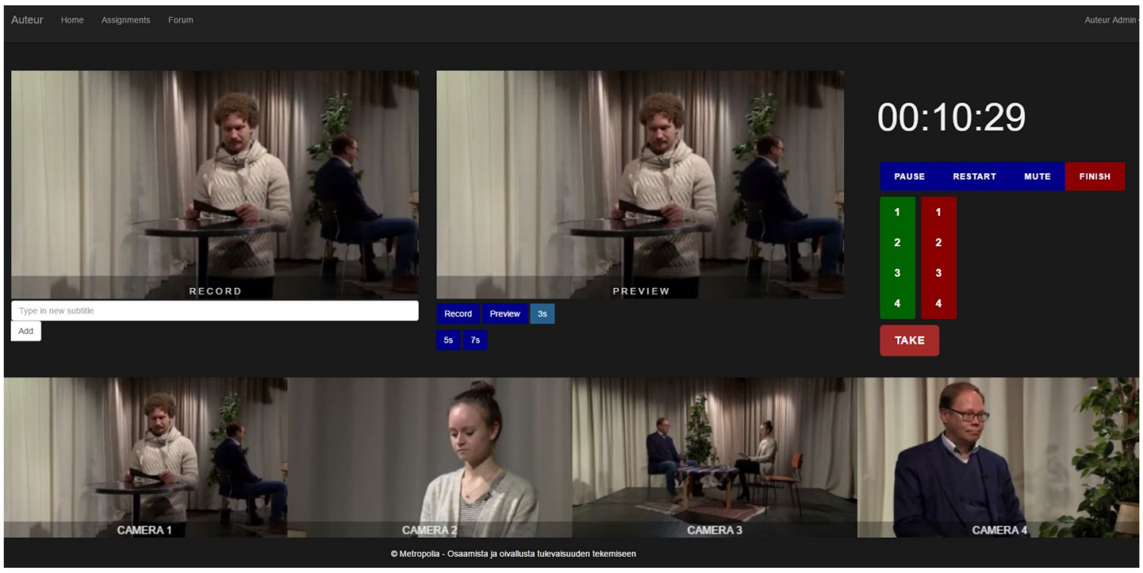


Figure 11. Simulator user interface, no camera switch.

The assignment is started by pressing the record button. Usually there is at least a 3-second window before any activity in the scene and thus student can adjust the record window view, during that time. The process of switching the views is relatively easy. Student has control only over the green buttons and apart from using the visual buttons, student can press 1,2,3 and 4 on his keyboard to select the cameras. The red “buttons” act only as indicators as for which camera is shown on the record screen.

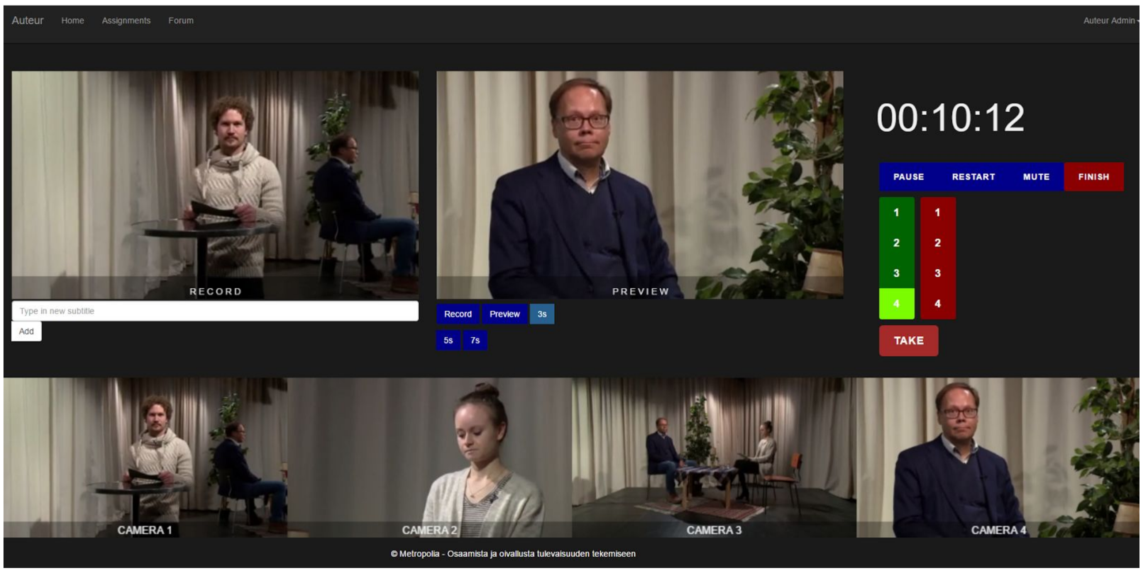


Figure 12. Simulator preview camera selected.(Screenshot from application)

As in figure 12, once the preview camera is selected, the next step for the student is to decide what he wants to do with this camera input. He can add subtitles, decide that he does not want the camera angle and switch to another camera or time the switch to the record screen.

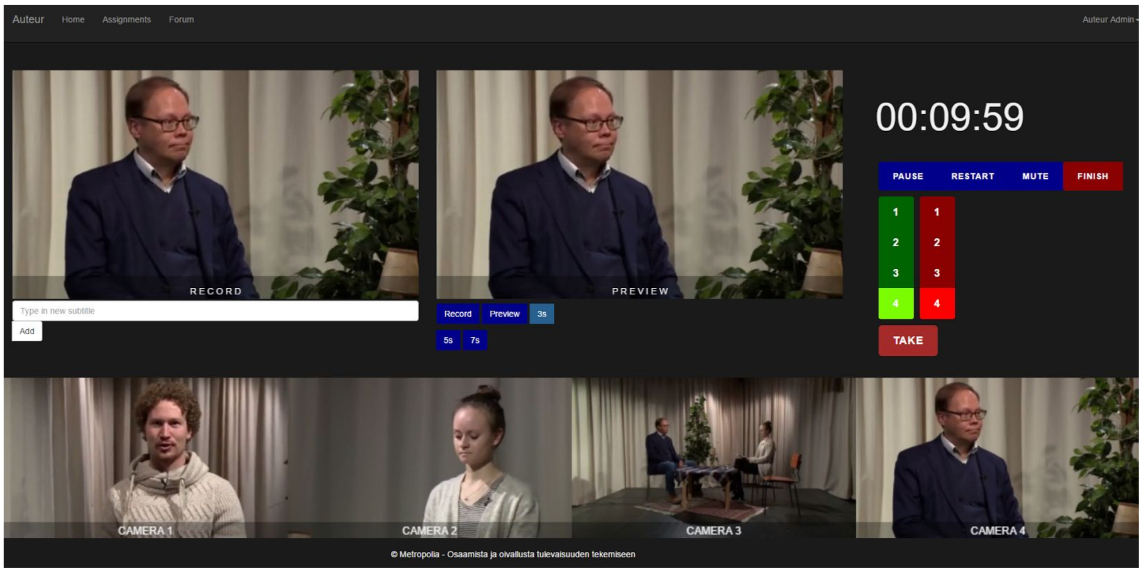


Figure 13. Simulator record view switching.(Screenshot from application)

The record window view is changed by pressing the Take button on the graphical user interface or Enter on the keyboard. The camera that is on the preview screen is then transferred to the record screen as shown in figure 13. The interface also has a timer, which helps student to understand how long the assignment will continue and if there is a script for the assignment that student has at hand, he can expect certain events happening in the scene at certain moments. Finally when student thinks that he finished the assignment, he can press the finish button, otherwise the assignment finishes once the time runs out. New button “View Result” will appear, which student can use to directly access his own result. In case student wants to view the result later, he can return to the assignment details view at any time and access it from there. Furthermore, if student wants he can delete the result, or in the middle of a simulator restart the assignment how many times he wants.

Teachers can upload the assignments through a corresponding interface. Moreover, the application also has a forum for establishing easy communication between students, teachers and developers.

5 Results

As a result we achieved a working prototype of television director simulator. Despite the end result being a prototype, the service is in a usable state and thus students as well as teachers can benefit from it on the television director course. Although the application is not currently used intensively in production (the application is going through single user tests now and then), we received positive feedback from Pekka Korvenoja and other teachers about the simulators' resemblance to the actual production studio, ease of use and technological aspects of the simulator, such as lack of delay between assignment execution and end result demonstration, camera switching and effective use of local storage on the backend. Compared to Youtube experiment "Choose your view" we managed to avoid audio stutters and any other delays between switches. However, in order for our application to be able to deliver 1080 pixels or higher quality video, the client should have a fast enough network connection and enough bandwidth to handle the stream.

The user tests for assessing the scalability have not yet been run. However, we assume it can easily handle at least 200 students at the same time considering all the logic behind the implementation. The different story is with the teachers and file uploading. Since the file upload considers also video editing, some rules have to be made on how many teachers can create their assignments at the same time. Video editing is a resource-heavy task and the system has not yet been tested for the upload pressure. Therefore, we assume that if more than three teachers would try upload their assignments at the same time, there is a risk that the server will hang up or the application will crash, because of over-usage of resources on the server side for the video editing task. The regulations for assignment creation can be defined on the application level and can have rules such as:

- Maximum number of assignments allowed to be created by one teacher.
- Maximum number of assignments allowed to be created by one teacher in one day, week or month.
- Restricted simultaneous assignment creation.

- Scheduled assignment creation - for each teacher he has his own day and or hour when the assignment can be created.

The rules are not limited to these options. Nevertheless, if there is a requirement for simultaneous assignment uploads, then the main components to improve in server will be CPU and RAM. CPU needs to have enough threads to accommodate each teacher and fast enough to execute the video editing tasks within reasonable time limits. In addition to CPU and RAM, local storage might need to be expanded from time to time, as number of files that each of the assignments will hold is four plus the amount of different qualities - four input videos for potential download feature implementation and combined assignment videos for each of the quality choice that the application has.

6 Potential future feature improvements

The multi-camera director simulator is still a prototype and therefore there are plenty of features to improve. Not only that, but there is also room for new features.

Graphical User Interface

The graphical user interface does not irritate the eye, but it definitely could be more beautiful and dynamic. Compared to modern web-applications it lacks in both beauty and design concepts, as it is very basic. Furthermore, we did not run any user testing to receive feedback on how comfortable the interface is, although the client was satisfied with the fact that the assignment page interface was similar to the one that the studio has.

Video upload

Video file upload is working, but from the user perspective it would be nice to see progress bar at least. Also there is no option for adding more than four videos, as the input amount is currently fixed and there is no distinction between cameras or inserts (temporary advertisements). All the inputs are treated as cameras from the show. For inserts there should be an option to play video of predefined length at some point of the assignment. Furthermore, it would be interesting if students could add their own inserts.

Authentication Security

For the user authentication the security has not been penetration tested. Currently the authentication is implemented with the help of Node.js module. The user passwords are hashed and stored in the same database, where the user assignments reside. Other aspects of the application have not gone through any penetration testing and therefore it is unknown if the application is secure enough or not. However, because the data is not very sensitive, the security aspect can be overlooked. Although if teachers are to use this prototype for organizing some sort of an exam, they should be aware that students might have a possibility for changing their end results, if the application proves to be unsecure.

The authentication handling could be transferred to another application with its own database and the multi-camera director simulator, could be then changed to fit the authentication model, which the third party application is using. There are services, such as Auth0 and Stormpath that supports OSS developers as well as commercial applications in this matter.

User Management

Teachers are currently admins in the system, so they can view all the users that there are in the system and add different students to certain student groups. The interface works, but it does not provide many features.

Assignments

Assignments could have additional features. The subtitles are mainly useful for displaying names of the show participants. The subtitles are in plain text and do not look too interesting. Some default graphics for displaying the subtitles could be added. Furthermore, as already mentioned in the file-upload section, there is no option for inserting the advertisements, nor any other static video content. Inserts are quite often used in talk-shows nowadays and it would be useful for the student to have some experience with this feature. Furthermore, shows also have some sort of music playing in the background at the start and the end of the show, sometimes even in the middle of the show. An option for adding custom audio tracks could also be added to the assignment features.

At first all these additional features might seem to complicate the assignment completion for one person, but since the assignments can be repeated before the submission, students could prepare themselves beforehand with all the additional materials and plans on when to insert them. As for the public talk shows and plays the camera switching happens often depends to some extent on the script according to which participants progress through the show.

Assessment template

The multi-camera director course teaches students that there are many rules about which camera to show to the public and when. Student should understand if the frame is good or bad when doing the assignment. This means that teacher could specify certain time-frames, when the student should show only one of the cameras from a certain selection, while also defining a time-frame when a certain camera should never be shown. This sort of assessment template could be implemented in order to speed up the process of grading the assignments from the teacher perspective. In case the template is improved even further, chances are that the assessment could be also automated, while teacher could give feedback on unique cases.

Live tasks

Something that we did not really try was the possibility of the live streaming tasks. The live streaming could be simulated by the fact that the pre-recorded videos are accessible only once and during a certain period of time. This kind of setup is good for making exams. However, true live streaming with ongoing video and audio synchronization needs more research. Of course there are existing studio solutions that implement the live streaming for the events, but these kind of solutions are usually offline in a sense that director has to be present at the event. Nonetheless, the multi-camera director has to be present at the studio in order to operate the streams. If there was a web-application that multi-camera director could use for handling real events, this could have potential to expand the bounds of live-streaming.

7 Conclusion

In conclusion, the project goal was to build a television director simulator that students could conveniently use at their home. As a result of the project the web-based television director simulator was produced where teachers can set up the simulator environment in the form of assignments that students would complete. Furthermore, since the application does not require a powerful computer for using it, students can do the assignments at any time they want from their personal computers. In addition students can choose to use even tablets or phones, as long as they have a fast network connection with enough bandwidth. Therefore, the required project goal was met.

The television director simulator allows to train future directors by simulating live broadcast events. Students can make a choice between multiple cameras to show to the auditorium while the end-result itself is recorded. Teachers on the other hand can set up the simulation and assess the results produced by students. Furthermore, since the application is web-based and not an offline desktop application, students do not have to download anything, which means that there is no time wasted on setting up an application from the students' perspective. Compared to existing solutions, the application helps more students to experience the television director's work in a smaller time window. Therefore, the application speeds up the training process of a multi-camera director.

The topic of interactive multi-camera streaming is quite complex. There are many application purposes and implementation approaches that were demonstrated at the beginning of the thesis. Finding the correct solutions for the application can require creative thinking from developers. The television director simulator that was produced as a result of the project is a perfect example of how different multi-camera applications can be. The simplicity of the produced solution does not mean its ineffectiveness, but vice versa, the method allows to save computer resources and at the same time provides good user-experience. Even though there are many aspects to improve like security, user management and graphical user interface, the project is a success.

References

1. Extron electronics. Modern stadiums. Extron electronics. Last update 2017. URL:http://www.extron.com/company/article.aspx?id=stadium. Accessed 3 April 2017.

2. Television studio. Wikipedia. Last update 5 January 2017 URL:https://en.wikipedia.org/wiki/Television_studio. Accessed 3 April 2017

3. Creative skillset. Television director responsibilities. URL:http://creativeskillset.org/job_roles/294_director_tv. Accessed 3 April 2017.

4. Zhong Zhang, Andrew Scanlon, Weihong Yin, Li Yu, Péter L. Venetianer. Video Surveillance using a Multi-Camera Tracking and Fusion System. ObjectVideo Inc. Last update 5 October 2008. URL:https://hal.inria.fr/inria-00326754/document. Accessed 3 April 2017

5. Xiaogang Wang. Intelligent multi-camera video surveillance: A review. Elsevier. Last update 20 July 2012. URL:http://www.sciencedirect.com/science/article/pii/S016786551200219X. Accessed 3 April 2017.

6. Mark D. Youtube “Choose your view” concept. Quora. Last update 6th September 2016. URL:https://www.quora.com/I-just-watched-a-YouTube-muti-angle-choose-your-view-video-Madilyn-Bailey-After-watching-the-video-I-had-many-questions-in-my-mind-and-want-to-know-bits-and-pieces-on-it-How-are-YouTube-multi-angle-videos-created. Accessed 3 April 2017.

7. Morningfrost. Production control room image. Last update 30 August 2008. URL:https://en.wikipedia.org/wiki/Production_control_room#/media/File:SKY_Sport24_PCR.jpg. Accessed 3 April 2017.

8. Napier Lopez. Experimental YouTube feature lets you choose from multiple camera angles. The next web. Last update 4 February 2015 URL:https://thenextweb.com/insider/2015/02/04/youtube-letting-choose-multiple-camera-angles-new-experiment/#.tnw_ntYeGpxE. Accessed 3 April 2017.

9. Peter Wiggins. YouTube testing user switchable multicam playback on videos. FCP.co. Last update 6 February 2015 URL:http://www.fcp.co/hardware-and-software/1599-youtube-testing-user-switchable-multicam-playback-on-videos. Accessed 3 April 2017.

10. Thomas L. Marquardt. ASP.NET thread based operation. Microsoft. Last update 20 July 2007. URL:https://blogs.msdn.microsoft.com/tmarq/2007/07/20/asp-net-thread-usage-on-iis-7-5-iis-7-0-and-iis-6-0/. Accessed 3 April 2017.

11. Benjamin Erb. Concurrent Programming for Scalable Web Architectures. Ulm University. Last update 20 April 2012. URL:http://berb.github.io/diploma-thesis/original/042_serverarch.html. Accessed 3 April 2017.

12. Transmission Control Protocol Specification. Information Sciences Institute University of Southern California 4676 Admiralty Way Marina del Rey, California. Last update September 1981. URL:https://www.ietf.org/rfc/rfc793.txt. Accessed 3 April 2017.

13. Packet loss. Wikipedia. Last update 25 April 2017. URL:https://en.wikipedia.org/wiki/Packet_loss. Accessed 3 April 2017.

14. MongoDB. Mongo and MySQL associations. MongoDB. Last update 2017. URL:https://www.mongodb.com/compare/mongodb-mysql. Accessed 3 April 2017.

15. Progressive Download. Wikipedia. Last update 7 October 2016. URL:https://en.wikipedia.org/wiki/Progressive_download. Accessed 3 April 2017.

16. H. Schulzrinne, Columbia U., A. Rao, Netscape, R. Lanphier. Real time streaming protocol specification. RealNetworks. Last update April 1988. URL:https://tools.ietf.org/html/rfc2326. Accessed 3 April 2017.

17. Adaptive streaming. Wikipedia. Last update 24 March 2017. URL:https://en.wikipedia.org/wiki/Adaptive_bitrate_streaming. Accessed 3 April 2017.