Abstract—This paper presents a web application framework, which uses sharing of eye-gaze information with different participants for multimedia analysis. Shared multimedia analysis is being used in fields such as Medical research, Intelligence, Web Analytics. Eye tracking aids these fields by providing Eye Gaze information so that participants are able to identify Areas of Interest of each other, which helps to get a better understanding of each other’s perspective. The framework focuses on a web interface which helps to analyze simultaneous playback of video among all the clients and displaying a dynamic heatmap of the gaze information to get a better control of visualizing gaze events.

I. INTRODUCTION

The advent of file editing and sharing platforms introduced many approaches and techniques in the domain of collaborative, real-time applications. These applications were directed to ease analysis and working. Eyetracking adds another layer of feature which can be shared among different participants on these kinds of platforms. Particularly, it involves collecting the gaze information of different users and sharing it, so that apart from sharing just screen, participants also have the information about each other’s gaze area via visualization that is being rendered on the screen.

Eye Tracking is performed using specialized hardware required to trace the eye gaze information. These devices are equipped with infrared cameras which are used to get the corneal reflections of the infrared light, which is emitted from the hardware and computer vision techniques are used to detect the center of the pupil. Together, this information is used to estimate the coordinates of the eyegaze position on the screen. In figure 1 we see that how the eyegaze information is estimated using mapping the vector from the center of the pupil to the position of infrared light being reflected on the pupil.

This reflected light on the pupil is used for getting a vector from center of the pupil to the reflected light on the cornea. Which is further mapped onto the screen to get the eye-gaze information. There are different types of Eye Trackers being used according to their utility, namely Head-Mounted eyetrackers and Remote eyetrackers. Head-Mounted eyetrackers record eyegaze information according to the first person view, where you can see the actual sight of the user and also the eye gaze information simultaneously. Remote eyetracker is installed away from the users, usually interfaced to a system, and then they track the eye gaze information. The difference between these hardware is, a person using head-mounted eyetracker is mobile and a person using a remote eyetracker is stable, so that remote eyetracker could detect eyes. But, the inherent technique used to track the eye gaze information is same.

Eye-Gaze information related to eye-tracking is related to many events namely Fixations, Saccades, Blinks, Smooth Pursuit, and vestibulo-ocular reflex.

Fixations are events in which a user dwells on specific areas of interest more than the specified threshold period of time. Saccades are the movement of eyes between those fixation events. A perfect example would be while reading any book, we see that our eyes move very smoothly over every line we read, but in reality, our eyes perform fixations i.e. jumps over every word we read, and the movement from one word to other between two fixations is a saccade. Blinks refer to closing and opening of the eyelid while viewing.

Smooth pursuit event occurs when eye locks on the movement of an object and follows it. This is different than saccades because it needs a reference object. Without the reference, eyes would just perform saccade. Vestibulo-ocular reflex is an event in which eye movement balances head movement while fixating on an object. These all events are very important when working with eyetracking. This paper mainly focuses on fixation and saccade events.

Visualizations play a pivotal role in displaying the eyegaze
information because without them it is nothing more than a set of coordinates. Various visualization techniques such as scatter plot, Saccade Pathways, heatmaps, Beehives, etc. are used to render the Eyegaze information. The choice of the visualization technique depends on the utility of the eyetracker results.

An eyetracker provides us with resourceful insights about eye movements and different events related to those movements. But, the only limitation of eyetracker is that it can only aid in knowing the eye gaze information about the user, but not the underlying cognitive information about that reason for moving eyegaze to that specific position.

This paper presents a web framework which shares eye-gaze information between all clients over a synchronized video playback. Additionally, users without eyetracker can also use the framework and all the clients can use the chat-room to communicate with each other. It collects raw gaze information and detects fixations. These fixations are stored and most recent fixations are rendered on the user interface. The Eye-Gaze information is rendered on the screen using dynamic heatmap, which moves according to the eye movements of a particular. Every user with or without eyetracker has a heatmap displayed on the list of current online users on the framework.

II. RELATED WORK

John et al., 2014[3] describe an application where real-time eye tracking information is shared between different users on different systems. Their application is intended to track the gaze of users to aid understanding on how images are being observed by different users. This system also facilitates identifying key areas of interests along with promoting faster analysis of images.

Fig. 2: Server-Client Architecture [3]

In figure 2 the architecture used in their application is a centralized server to communicate between different systems and display gaze paths. The 3-tier architecture which uses eye trackers as end nodes to collect locations of the viewing paths for various clients based on their IP addresses. Client systems then communicate with the centralized server to share their location data and receive data from other clients. The application framework then visualizes all the information received on the client system. This removes the dependency on the server thereby allowing the application to be scalable. This paper adopts a similar strategy and architecture for deploying the client-server based system to track gaze paths of different users on synchronously streaming videos.

This framework created in Java provides a simple, user-friendly GUI, to elaborate and visualize the gaze paths of various users and highlight them. The testing procedure followed in this paper demonstrated the advantages of collaborative eye tracking to solve differences in images with help of multiple users. Hence this paper provided a premise to further research approaches and solutions to collaborative action between multiple users at discrete locations.

Venugopal et al., 2016 [4] discussed the application of eye tracking frameworks in the domain of medical, marketing and assistive technologies along with enhancing digital platforms. This paper explored how augmenting eye tracking capabilities to devices or consoles enhance its usability and feasibility for analysis. This paper also identifies how prediction models and statistical learning can be used to analyze how images are viewed. This allows better understanding how different users perceive media and how analysis and visualization of gaze paths augments creation of interactive media.

The architecture used for creating the application is entirely client-based system. The architecture consists of a storage unit which is used to perform analysis of user behavior. This behavior is retrieved from eye tracking devices mounted on the client system.

Malčík et al., 2014 [5], provides an understanding of how information is retrieved in the visual form is researched. This paper provides the premise on how visual patterns can be analyzed to highlight how textual data is perceived by different users. It also discusses how different platforms of media are viewed and perceived differently by users.

Understanding how various metrics with respect to eye movement, pupil location, etc can be analyzed is an important step in how visual information is collected from different media. These metrics provide us the correlation on how different approaches can be used to make a particular media more readable. It also lays the groundwork for devices and applications that are used in this paper to be introduced modern day devices to augment learning. This can also strengthen the understanding how cognitively materials are viewed in different media. The various tests in this paper allow real-time reporting of behavior and provide analysis and scenarios which challenge users to perform discretely to various visual stimuli. Such techniques are extended in this paper to enhance the functionality of the eye tracker application as well as provide criteria for assessment and research.

In Ozkan & Ulutes., 2016 [6], the application of eye tracking is extended to handle and augment industrial processes like inspection and quality assurance. This paper demonstrates how eye tracking can be used to augment human processes at the production level and lays the foundation to research in assistive technologies and applications.

Singh et al., 2016 [7] introduces the application of eye tracking is used to augment surveillance along with deepening
our understanding on, how visual information is conceived to make decisions based on behavior. This paper also sheds light on how visual media can be enhanced to be more informative or palatable by understanding the learning patterns of different users.

The architecture in this research involves understanding and measuring the key metrics of eye movements and cognitive processes to map the variance in view paths or area of interests for given visual media. This paper also combines the use of tracking brain waves along with view paths to highlight how information is processed and decisions are made.

III. SYSTEM DESIGN

The Web Framework discussed in this paper has a standard server-client architecture in which different clients connected to the server and the data was being broadcasted from each client to every other client connected to the framework. The Server was developed using Express NodeJS framework and rather than hosting front-end for the server on the different system it was developed as a Server-Side rendered web application.

Every client needed to have a specialized hardware for eyetracking purposes, which was attached to the system being used for communicating with the server, and then send the data to the server. The Hardware was provided by SensoMotoric Instruments, which is a Germany based company specializing in developing eyetrackers.

A. Hardware

The device used for eyetracking was SensoMotoric Instruments RED250 Mobile eyetracker. It is Remote eyetracking devices, which is attached on to the client’s system and records the eye-gaze information. The technical specifications of eyetracker are as follows:

<table>
<thead>
<tr>
<th>Specifications</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling rate</td>
<td>60, 120 and 250Hz</td>
</tr>
<tr>
<td>Gaze position accuracy</td>
<td>0.4</td>
</tr>
<tr>
<td>Operating distance</td>
<td>50-80cm</td>
</tr>
<tr>
<td>Tracking range</td>
<td>32 x 21cm at 60cm</td>
</tr>
<tr>
<td>Dimensions (w x h x d)</td>
<td>24 x 2.7 x 3cm</td>
</tr>
<tr>
<td>Weight</td>
<td>175g</td>
</tr>
</tbody>
</table>

Our understanding on, how visual information is conceived to make decisions based on behavior. This paper also sheds light on how visual media can be enhanced to be more informative or palatable by understanding the learning patterns of different users.

The architecture in this research involves understanding and measuring the key metrics of eye movements and cognitive processes to map the variance in view paths or area of interests for given visual media. This paper also combines the use of tracking brain waves along with view paths to highlight how information is processed and decisions are made.

III. SYSTEM DESIGN

The Web Framework discussed in this paper has a standard server-client architecture in which different clients connected to the server and the data was being broadcasted from each client to every other client connected to the framework. The Server was developed using Express NodeJS framework and rather than hosting front-end for the server on the different system it was developed as a Server-Side rendered web application.

Every client needed to have a specialized hardware for eyetracking purposes, which was attached to the system being used for communicating with the server, and then send the data to the server. The Hardware was provided by SensoMotoric Instruments, which is a Germany based company specializing in developing eyetrackers.

A. Hardware

The device used for eyetracking was SensoMotoric Instruments RED250 Mobile eyetracker. It is Remote eyetracking devices, which is attached on to the client’s system and records the eye-gaze information. The technical specifications of eyetracker are as follows:

<table>
<thead>
<tr>
<th>Specifications</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling rate</td>
<td>60, 120 and 250Hz</td>
</tr>
<tr>
<td>Gaze position accuracy</td>
<td>0.4</td>
</tr>
<tr>
<td>Operating distance</td>
<td>50-80cm</td>
</tr>
<tr>
<td>Tracking range</td>
<td>32 x 21cm at 60cm</td>
</tr>
<tr>
<td>Dimensions (w x h x d)</td>
<td>24 x 2.7 x 3cm</td>
</tr>
<tr>
<td>Weight</td>
<td>175g</td>
</tr>
</tbody>
</table>

TABLE I: Eyetracker Specifications. Courtesy:[8]

The eyetracker is setup from the user’s sight. To send the eyetracker gaze information to NodeJS server, Python Flask server and PyGaze library[9] was used. The features which were used on the Pygaze library were built on top of the Software development Kit Provided by SensoMotoric Instruments. The Pygaze library had methods which were able to return the coordinates of the eye-gaze information from the display screen and any eyetracking event occurrences such as Fixations or Saccades.

The Client-Side contained a python script, which sent the Eye-Gaze information to the NodeJS server using Python Sockets library and a Python Flask Server to Run that script. Basically, the setup was to run the Python Flask server on the client system and wait for the request from the NodeJS side server side to execute the script, which in turn sent the coordinates to the server. The eyetracker information for a particular user was not directly shown on the server side rendered page, i.e. first the information was sent to NodeJS server running, which is waiting for the response from the NodeJS server. The GET request from NodeJS server is triggered if any user accesses the server-side rendered page. As soon as that happens, the flask server on client-side executes a python script. This Python Script connects to the IViewRED server using Pygaze Python Package [9], receives the data and sends it to the NodeJS server central server using Python Sockets library. NodeJS server receives the data from all the clients and sends diverts it back to all the other clients.

This influx of data on the server is very high as a single eyetracker can emit almost 5000-1000 co-ordinates every 2-3 seconds. Therefore, client-side has to delay the data being sent to the server by almost 100ms. Other than eye-gaze data, clients also emit their Youtube player data i.e. player timings, which is compared with the host and then synchronized for individual clients. Additionally, chat room and user status are also displayed on the client-side. All this is Synchronization of data between different clients is handled by Socket.io, JavaScript Library.

C. Client-Side

On the Client-side, windows laptop was equipped with a SensoMotoric eyetracker which is attached below the screen using a magnetic mounting bracket. Additionally, SensoMotoric Instruments a has also Provided a software i.e. IViewRED, which helps in setting up the eyetracker on the client-side. On the setup screen, IViewRED demands display screen height and width and also the inclination at which the eyetracker is setup from the user’s eyesight.

Other than being the hardware side for the overall framework, client-side also had to send eye-gaze position coordinates of the display screen to the server. The communication between server and client was done using JavaScript Web-socket library Socket.io. Web sockets are known for fast and reliable delivery of data over the network between any systems. This Library is available as client-side Library and as a Server-Side library as well. This was the motivation to keep the overall web application as Server-Side rendered.

To send the eyetracker gaze information to NodeJS server, Python Flask server and PyGaze library[9] was used. The features which were used on the Pygaze library were built on top of the Software development Kit Provided by SensoMotoric Instruments. The Pygaze library had methods which were able to return the coordinates of the eye-gaze information from the display screen and any eyetracking event occurrences such as Fixations or Saccades.

The Client-Side contained a python script, which sent the Eye-Gaze information to the NodeJS server using Python Sockets library and a Python Flask Server to Run that script. Basically, the setup was to run the Python Flask server on the client system and wait for the request from the NodeJS side server side to execute the script, which in turn sent the coordinates to the server. The eyetracker information for a particular user was not directly shown on the server side rendered page, i.e. first the information was sent to NodeJS server running, which is waiting for the response from the NodeJS server. The GET request from NodeJS server is triggered if any user accesses the server-side rendered page. As soon as that happens, the flask server on client-side executes a python script. This Python Script connects to the IViewRED server using Pygaze Python Package [9], receives the data and sends it to the NodeJS server central server using Python Sockets library. NodeJS server receives the data from all the clients and sends diverts it back to all the other clients.

This influx of data on the server is very high as a single eyetracker can emit almost 5000-1000 co-ordinates every 2-3 seconds. Therefore, client-side has to delay the data being sent to the server by almost 100ms. Other than eye-gaze data, clients also emit their Youtube player data i.e. player timings, which is compared with the host and then synchronized for individual clients. Additionally, chat room and user status are also displayed on the client-side. All this is Synchronization of data between different clients is handled by Socket.io, JavaScript Library.
Fig. 3: System Architecture

server and then to the browser on which the user was accessing the server-side rendered page.

D. Server-Side

The server was hosted on Raspberry PI 3 Model B. It was developed using Express NodeJS web application framework. The application was accessible as a server-side rendered page to all the clients. The server is able to detect if the client has an eyetracker connected or not. The Server-Side rendered page displays the Visualizations of the eyetracker using a dynamic heatmap, which is rendered on the screen according to last few co-ordinates form the eye-gaze data, which the client can set on the front-end.

Main objectives of the server are to:
- Send the GET request to Python-Flask server on the client side to trigger script which sends eye-gaze data back to the server.
- To parse the data from received from all the clients and send it back to all clients.

Whenever any client connects to the server, an HTML page is rendered. This rendered page contains three different components. A Youtube Video frame, chat window and a list of online users with an option to disable the heatmap for a particular user and to know if a user has an eyetracker or not. The Youtube video is synchronized on all the clients, and the first person to connect to the server is the host for that session. The host has the control for playback of the videos on all the clients, i.e. the seek-bar on the Youtube player of all the clients is dependent on the position of seek-bar on the host’s machine.

The visualizations displayed on the screen are rendered using browser-based local Javascript array. The array stores the coordinates of each users eye-sight according to the unique socket ID of the client connected to the server. heatmaps are created using n last entries in the array, where n is set by the user through front-end. For Updating Dynamic content on the server-side rendered page jQuery and JavaScript was used. BootStrap was used to render HTML elements on the page. Both, Client-side and the Server-side had Socket.IO library to communicate with each other. So basically, every client-side rendered page has a unique web socket object to transfer data between each other.

E. User Interface

The User-Interface is designed using HTML, CSS, JQuery and rendered on top of Express NodeJS Framework. On the first page, page prompts the user-name of the client to be displayed on the next screen with Video and Chat-room. While the user enters the name, the NodeJS server sends GET request to Flask server on client-side and Detects whether an eyetracker is connected to the system or not. If the user is first to connect to the server, he acts as the host for that session. When all the users have connected to the Session, the host hits the play button on the Youtube Video player, and the video simultaneously starts playing on all the connected machines.

As soon as the client with an eyetracker provides the user-name and transitions to next page, A dynamic heat-map is rendered on the screen. All the users are provided with different colored heatmaps so that users can easily identify the eye-gaze information for a particular user. Further, the host controls the video playback, and users can communicate with each other using Chat-room.

In figure 4, We have User status Area, Chat-box, and Youtube video player. The User status area displays which users are connected and if they have an eyetracker or not. On the Youtube player, a dynamic heatmap is rendered. In figure 5, User also has the option to render trails of heatmap...
i.e. which represents data for the range of 2-3 seconds, rather than instantaneous data.

IV. RESULT

The evaluation of the web framework is based on the increase in latency based on the number of clients. Total 3 different evaluation were performed, which were based on the number of eyetrackers in the system. In each iteration of evaluation, five clients gradually connected to the system and average latency across all the clients was calculated. The experiment was performed with 2 eyetrackers and a cursor movement on one of the systems was used as a dummy third eyetracker. In figure 6, it is seen that as the number of eyetracker increases, the slope of the line increases for that particular set of points. Which is a clear indication that as number of eyetrackers increase, latency is bound to increase and hence we see a delay in rendering of Heat-map.
V. CONCLUSION AND FUTURE WORK

The framework was successfully able to share eye-gaze information amongst connected remote clients. The objective of the framework was to synchronize real-time eye-gaze data effectively over a video file, which can help speed up analysis. This was achieved by the application, but it still has some limitations, as such with the number of clients connected to the system. The latency can be reduced by optimizing server’s handling of the data being received by the eyetrackers, which is the main cause of the latency. The eyetracker emits several co-ordinates in such small amount of time that it becomes difficult for the server to parse the data received effectively. As seen in the Results section, The latency slope is proportional to the number of eyetrackers in the system. Optimization on the server-side to handle a large amount of data and then redirecting data to all the other clients, along with Youtube video being synchronized across all the clients is the way to get minimal latency on the Framework.

Additional features such as voice communication between clients and choice of color coding for rendered heat-map can be added to the existing framework. Any other visualizations can also be used to display eye-gaze information on the screen.

REFERENCES