Virtual Cinema

Hojae Yoon (hy2714)
Shrihan Pasikanti (sp3921)
Sankalp Apharande (spa2138)
Yukti Khurana (yk2950)

1. Introduction

When was the last time you visited a movie theater? Virtual cinema brings the cinema experience to your room with all the intimate interactions at the movies between you and your loved ones. Our web app allows hosting a *virtual watch party*, similar to the popular Netflix Party chrome extension, but with more features for a cherishable experience. Upto 4 users can join a room to stream together.

Here friends can watch together the videos from websites which allow video embedding – YouTube, Vimeo, Streamable, and even from Facebook! For an immersive experience, users can send reactions via the reaction feature, send messages, and watch everyone's video feed in real time.

Our project **aims to provide quality video watching experiences during the pandemic on top of a scalable cloud infrastructure**.

2. Problem Statement and Approach

***Problem Statement*:**

Our product helps solve the problem of creating intimate experiences within a room of close people. This is a much needed solution especially at this time where people cannot go out due to the pandemic and everything more-or-less moved onto virtual grounds. After conducting focus group interviews of friends and family, we've discovered that the pandemic rendered activities and interactions that were more natural in offline settings to be more acceptable online. One of them was video conferencing (instead of face to face meetings).

Diving deeper into the video conferencing experience, we discovered that most video call services (such as zoom or google hangouts) were one directional. A single user shares their screen, other participants interact during the presentation. Video conferencing apps did not support activities where all users collectively engaged in an activity such as playing video games together or watching movies together. Virtual Cinema can also be understood as an attempt to transform video conferencing into a truly bi-directional and engaging experience. We achieve this by harnessing the power of cloud computing and websockets.

***Approach*:**
We have developed a web application which runs on a browser. A brief walk-through of our application is as follows:

a. A host sets up a link to a video and invites family, friends, or their loved ones to join. The host can invite users via SMS or email and set a particular time to start the video (similar to scheduled zoom meetings). By default, we open a *Socket synced real time video*

player. The video continues to play from when the users joined – just like what happens when you hop onto a YouTube live.

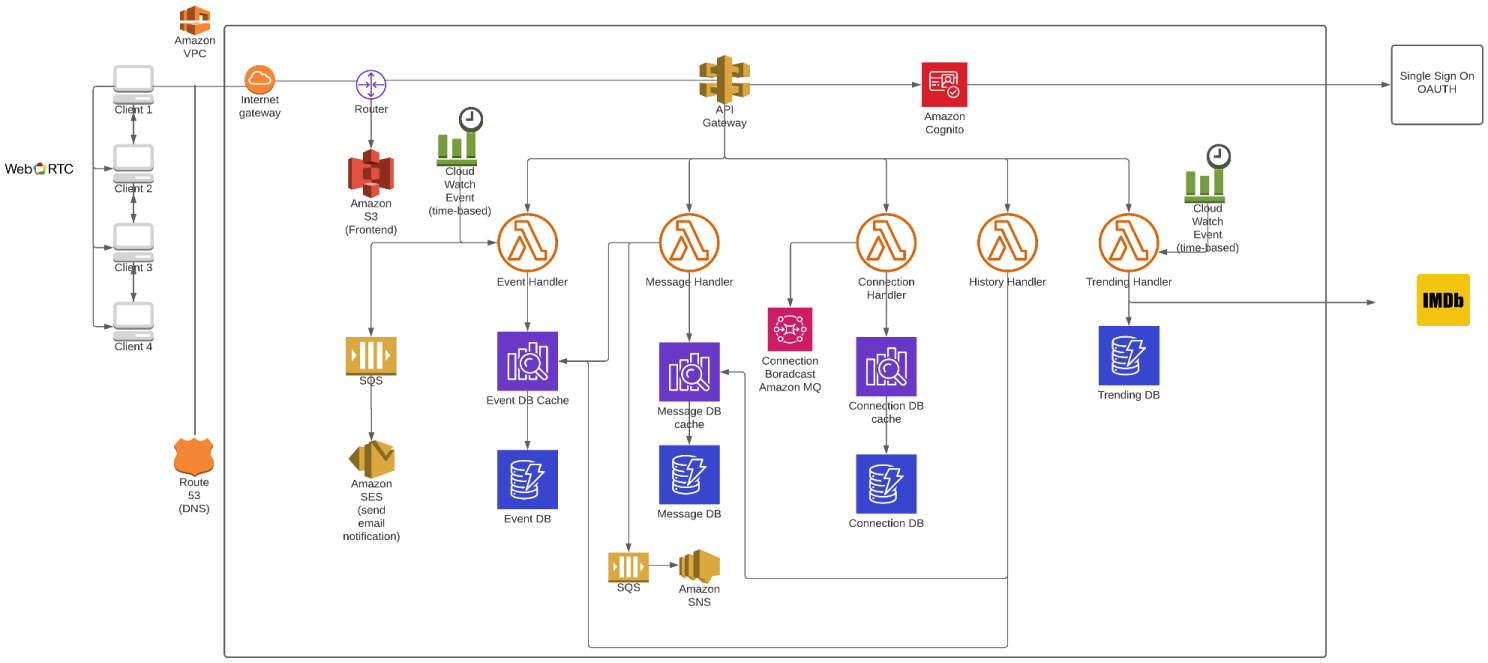
b. When the watch party is started – users can type into the chat box or throw up some reactions – just like a live chat in *YouTube live* video.
c. Users will be able to control the playback and speed of the video, which will be automatically synced.
d. Users also can chat over voice, similar to the discord VoIP feature, to simulate the experience of laughing or being terrified at particular scenes. They can also text each other via the message box.
e. Just like a subtitle file (e.g. .srt file extension), which we often see for movies, which store text for a video accompanied by its timestamps; we similarly store the entire message conversation log of all the friends in a room in a file. The file will be saved in the theatre room's log, to be replayed at a later point of time for friends to relinquish their memories!

3. Architecture and APIs

An Event is an atomic unit used to create a single instance of a watch party. Below is a list of all APIs implemented and our architecture diagram.

Lambda	Description	Method	Route
Event	Get One Event	GET	/event/{id}
	Get All User Events	GET	/event/
	Create Event	POST	/event/{new_id}
	Update Event	PUT	/event/{id}
	Delete Event	DELETE	/event/{id}
Message	Get messages for event	GET	/message/{msg_id}
	Publish message	POST	/message/{msg_id}
Connection	Get all user socket addresses for event	GET	/connection/{event_id}
	Publish client socket address for event	POST	/connection/{event_id}
	Update client socket address for event	PUT	/connection/{event_id}
	Remove client socket address for event	DELETE	/connection/{event_id}
History	Get single session history	GET	/history/{session_id}
	Get last 10 sessions' history	GET	/history/{no_sessions}
	publish history to db if user prompts	POST	/history/{session_id}
	Delete oldest 10 session's history	DELETE	/history/{no_to_delete}
	Delete a particular user's history	DELETE	/history/{user_id}

Account	Get a user account on login	GET	/auth/{account_id}
	Save a user account on registration with the service	POST	/auth
	update account details when user updates profile	PUT	/auth/{user_id}
	delete account on user demand	DELETE	/auth/{user_id}



4. Design Details and Code Structure

The design & wireframing of this application has first been done on Figma to ensure easy conversion to code. The codebase has been eloquently developed in a microservice fashion in contrast with the much more difficult-to-maintain monolithic architecture. We have a total of 6 pages in the UI each catering to its purpose. The codebase has been split into *two repositories:*

A frontend repository:

This base primarily acts as the client-facing interface. All the development tools/frameworks are state of the art and used in real industrial settings. It is primarily implemented in ReactJS and SASS. Instead of going on with the routine HTML/CSS framework, we used Sass: Syntactically Awesome Style Sheets in our framework which compiles to CSS code. It is both easy and maintainable for managing existing code and implementing new features.

We can run the clients either with *yarn / node*, both of which are really good at managing, resolving, and downloading all dependencies for a project.

Code Structure:
This repository has initially been set up with React-Bootstrap. Therefore, the general structure of the project is as created by the npm package manager. We are using *Axios* for API calls in the frontend.

A backend repository which:

Mostly acts as an aggregator for all of the code of the components hosted in AWS. It includes, but is not limited to API Gateway, lambda functions. We have hosted the application in an s3 bucket after buying a domain at hojaeyoon.com. We have implemented state of the art security mechanisms by implementing SSL/TLS certificates in the domain. This was achieved by attaching an SSL certificate to CloudFront (a CDN), that was synced with an S3 bucket that contained static React code. Doing this was needful as our application handles heavy video streams & messages via socket.io's connection handler, therefore it is necessary to provide our users with a secure platform to connect.

The APIGateway folder holds the SDK & the swagger file used to generate the SDK. It handles all the API requests like fetching user information, publishing messages to the room, fetching an event's history, scheduling an event among others.
The LambdaFunctions hosts all of the functions each of which serve a specific purpose. It includes the *AuthorizationHandlerLambda, ConnectionHandlerLambda, EventHandlerLambda, HistoryHandlerLambda & MessageHandlerLambda* among others:

AuthorizationHandlerLambda: Used for getting user information, authentication and storing it in a DynamoDB table.
ConnectionHandlerLambda: Responsible for connecting the clients together via broadcasting IPs. It provides an underpinning for Video player, reactions and messages synchronization.
EventHandlerLambda: Fetch all the events for a specific user, fetch all the metadata for a specific event created in the past, and store data pertaining to a newly created event in a DynamoDB table.
MessageHandlerLambda: Fetch all the messages for an event, fetch a message via its 'messageID', and push the messages to a *SQS queue* to be eventually stored in a DynamoDB table.

AWS Components:

The NodeJS server is hosted on an *nginx web server* which runs on an *ec2 instance* to cater to the clients connecting to it, and serving as a broadcaster to make sure all other clients' in a watch party know the socket connection object of the other clients. We have chosen nginx as it is designed for maximum performance, stability, its light-weight resource utilization and its ability to scale easily on minimal hardware.

To cater to users around the world, we have used *Amazon CloudFront* as our Content Delivery Network (CDN). Each time a dev change is pushed, we have a script to invalidate the cache stored in the CDN, so the change is reflected to all the users served by the CDN.

AWS Cognito is used for authentication of a user into the system/registration of a new user, using *user pools.* New user registration is done by having the user confirm their signup email address which is done using *Amazon SES*. All of these details are stored in a *DynamoDB* table. This table also stores event history, message history among other information. *Jwt tokens* are used as bearer tokens for authorization.

5. Results and Features Implemented

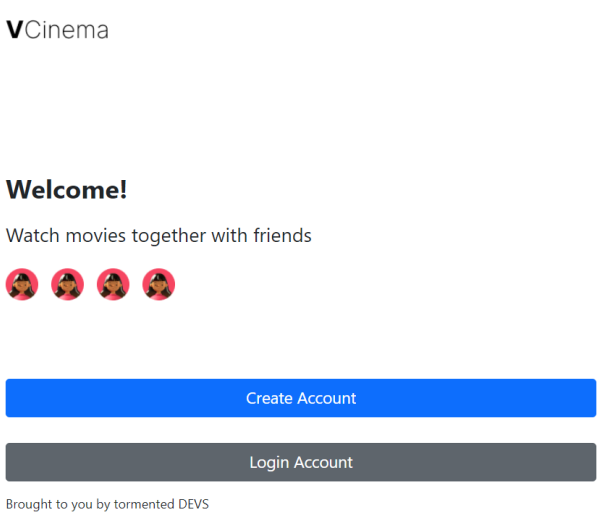
The presentation deck is linked here

The final demo video is linked here

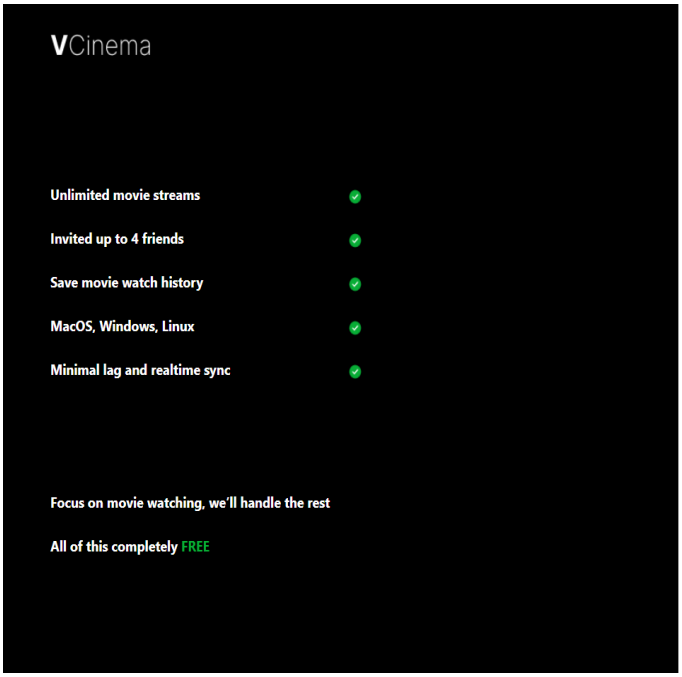
VCinema Application link: vc.hojaeyoon.com

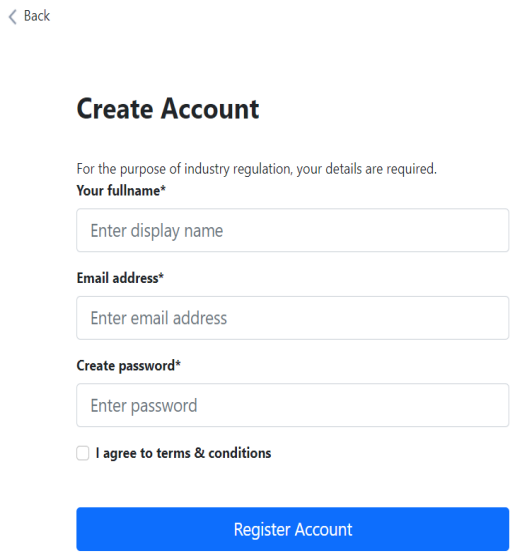
The VCinema Application created using React supports all kinds of devices like laptops, tablets, or smartphones. Following is the detailed information about the features our application supports for users -

Detailed app workflow:



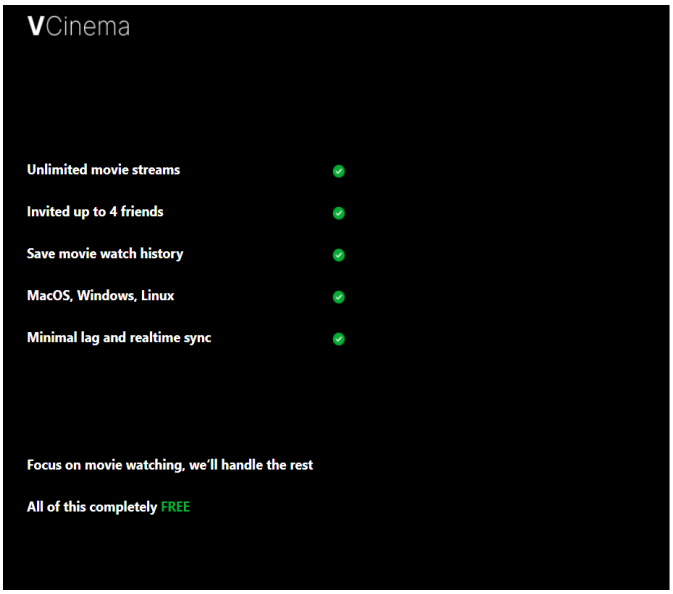
The Welcome Page - The welcome page of the application gives users the option to either create an account if they are new to the application or log in to their existing account so that they can keep track of their previous activity.

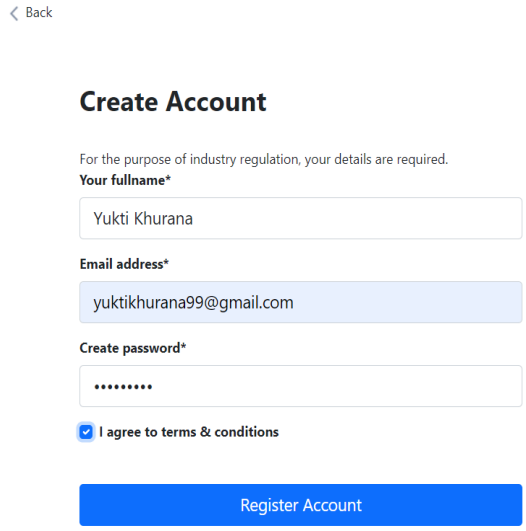


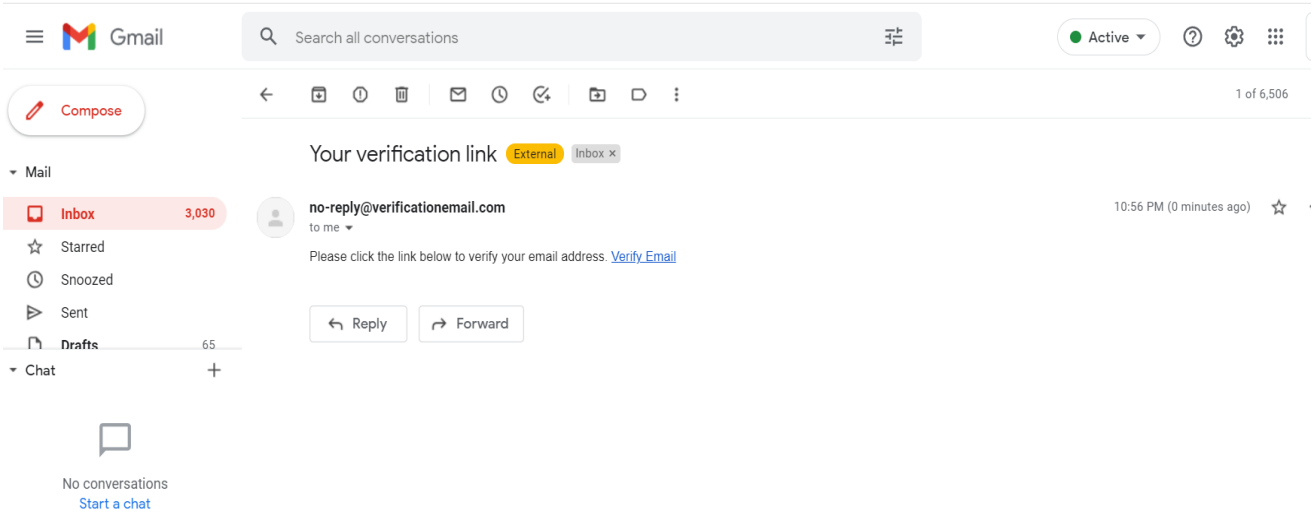


Create Account Page - Clicking the "Create Account" button on the Welcome Page takes the user to the Account Creation page where the user can enter their details to get associated with

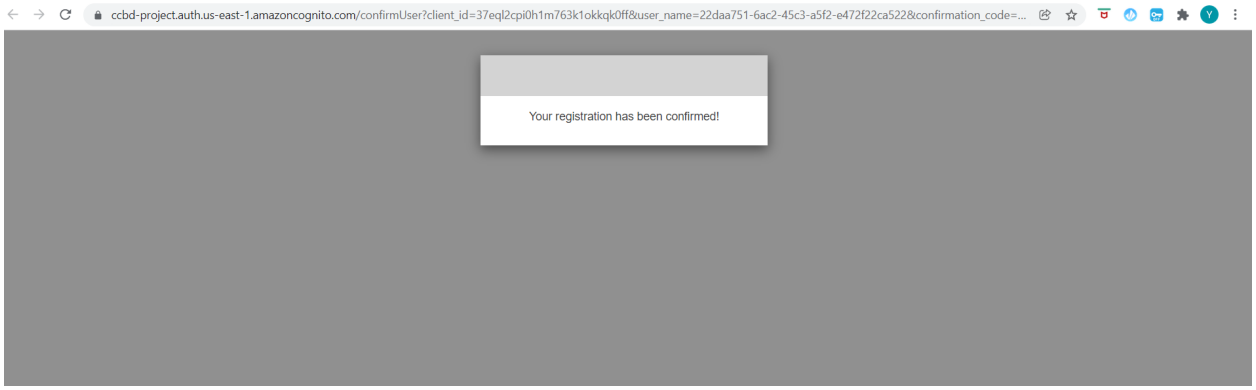
our application. This process is very simple for user convenience and requires users to only enter the name that they want their account to display whenever they log in, their email address so that we can verify their identity and a password to protect the privacy of each customer. After they agree to our terms and conditions, they can click on the “Register Account” button and their new account will be automatically created within seconds.



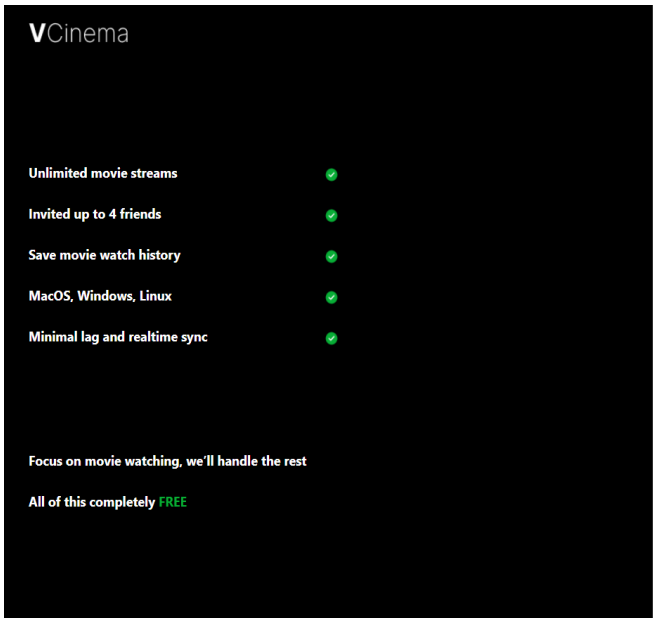


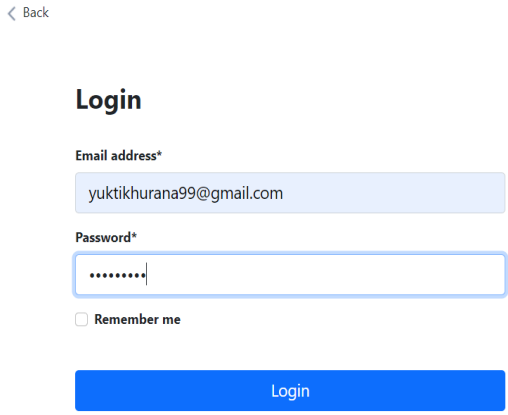


For security reasons, a verification link is emailed to the user on the given email address to verify that the email exists. The user needs to click on the “**Verify Email**” link obtained in the inbox, as shown in the images above to activate their user account.



After they do so, the “successful registration” message will be displayed on the screen.

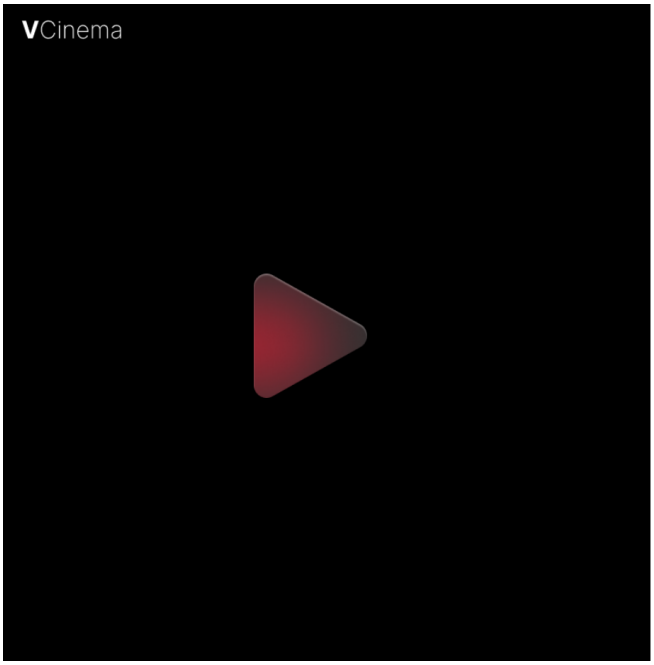


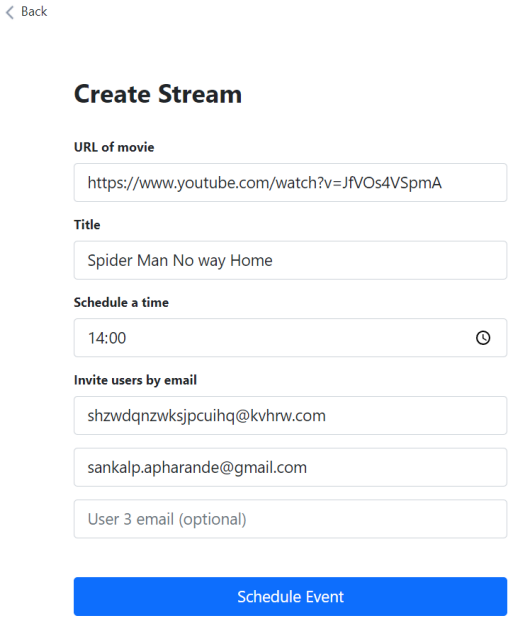


Now, the users will be redirected to the Login Page.

Login Page - After successful account creation or on clicking the “Login” button, the user is redirected to the Login page where they have to enter their correct email and password registered with the VCinema Application. If the login email and password are correct, AWS Cognito will authorize the user, assign a unique JWT token and let the user access our application.

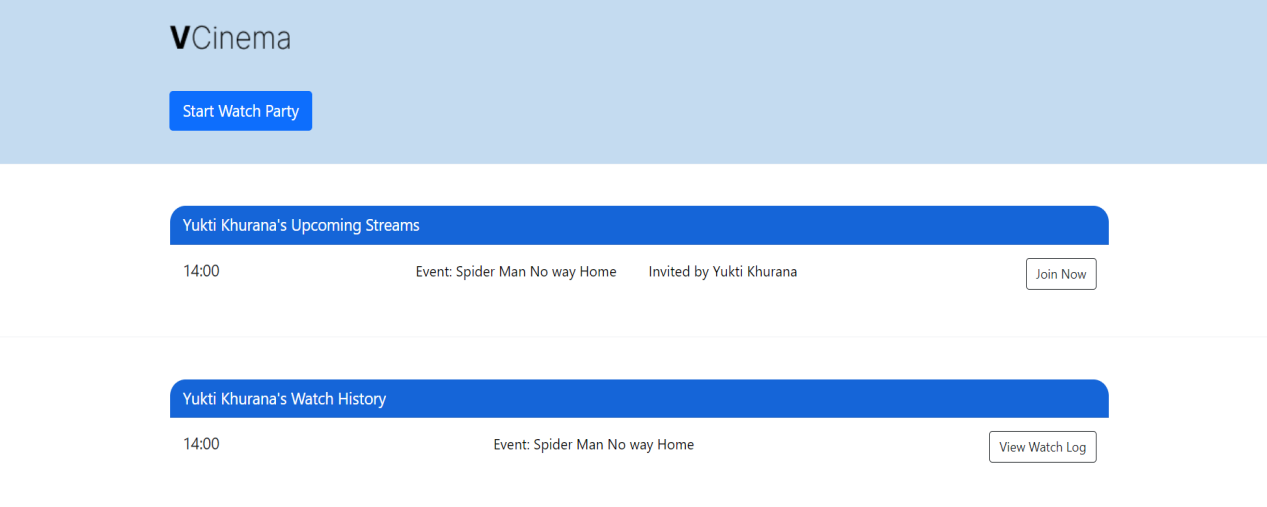
User Dashboard - On successful login, the user can see their application dashboard where they can view their upcoming watch streams/ watch parties that they can join. They can also see their "watch history" i.e. the streams that they have already attended. Users can also create a new stream and invite their Friends, Family or loved ones to watch any video together.





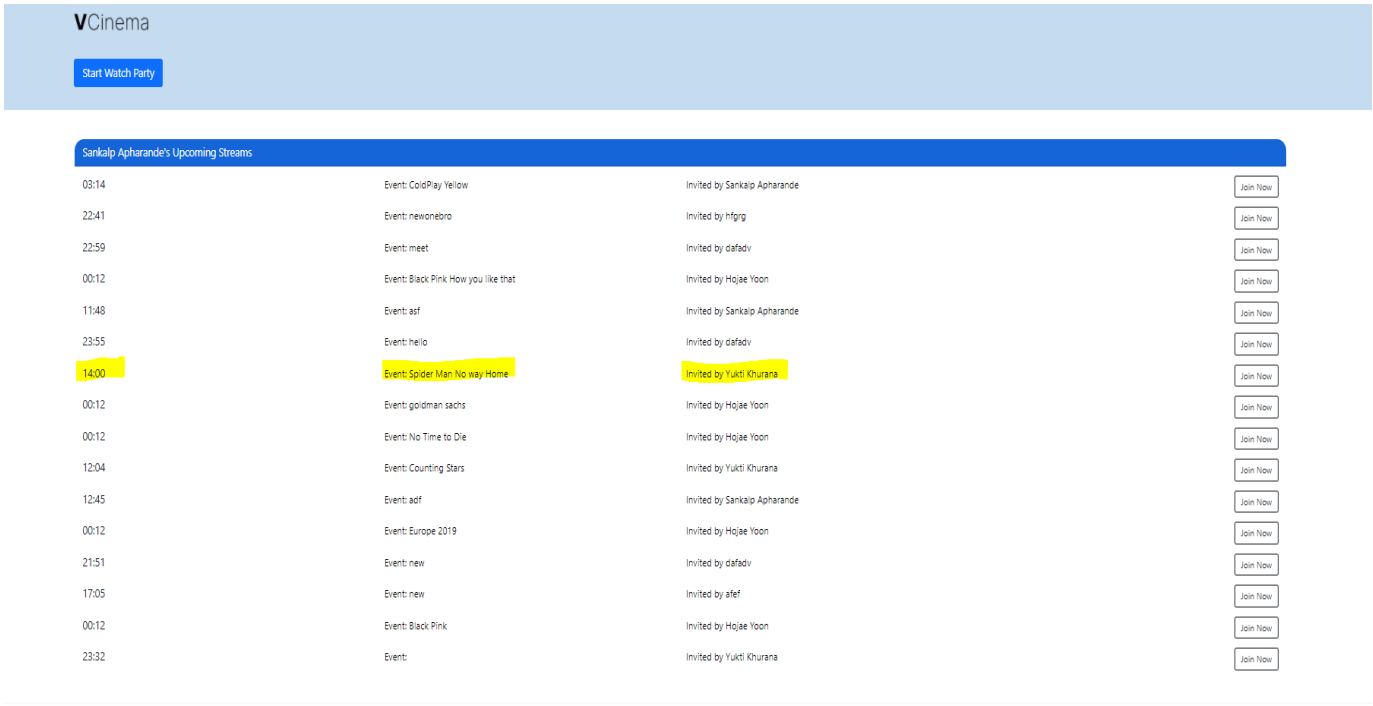
Create Watch Party Stream - On clicking on the "Start Watch Party" button, the user is redirected to the Create Stream Page of the application where they can create a new event

according to their preferences. They need to enter the url/link of any video/movie they want to watch with their friends and add a title to it so that other invited users can easily distinguish it from other events. The user then can enter any time at which they want to schedule the Streaming experience followed by the email ids of all the users they want to watch the video/movie with. They can enter a maximum of 3 users other than themselves to ensure an intimate movie/video watching experience. After entering all the required information, they can click on the “Schedule Event” button so the invite emails are sent to all users whose emails the user entered. This way all users are informed that their friend wants to watch a video/movie with them.



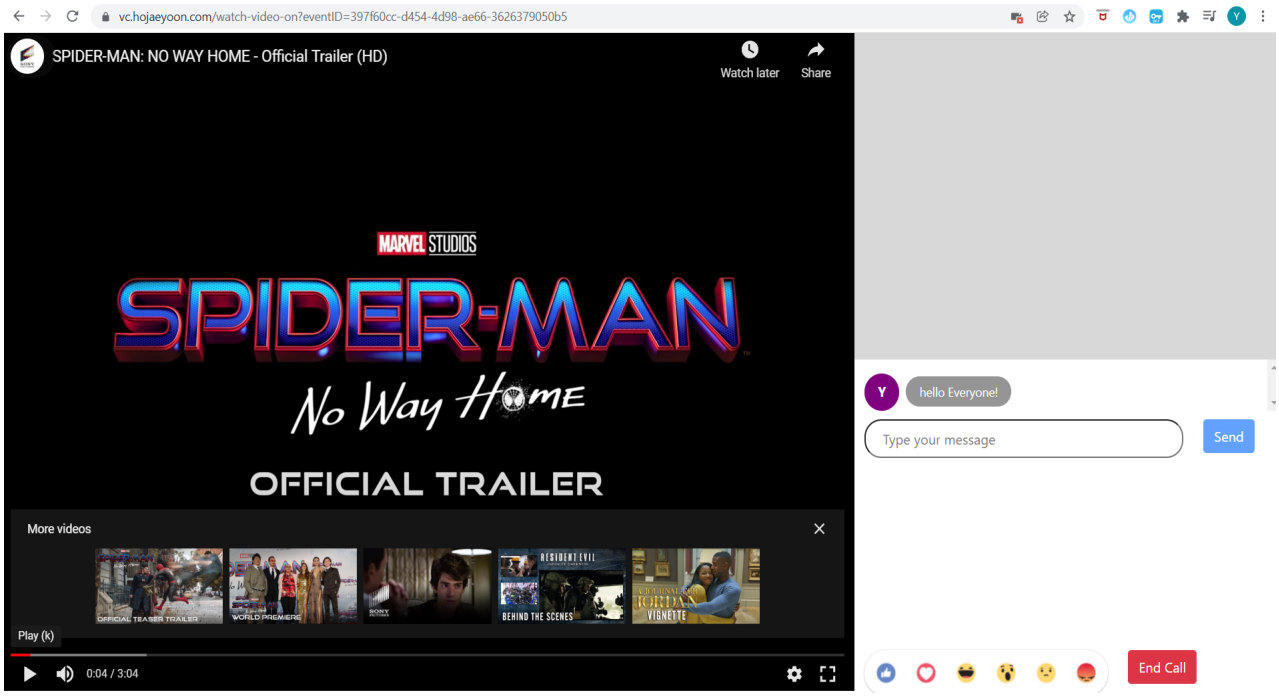
The newly created stream will be displayed on the user dashboard page as shown in the image above. This stream can now be joined by the user and all the other people who have been invited.

Upcoming Streams - As shown in the application screenshot below, the invited user’s dashboard now has a new event under the “Upcoming streams” option. It clearly mentions the time the event is scheduled to take place, the title, and the user who created the watch event. The “Join Now” button has been provided for all the users to join the stream and watch the video with their friends and family.

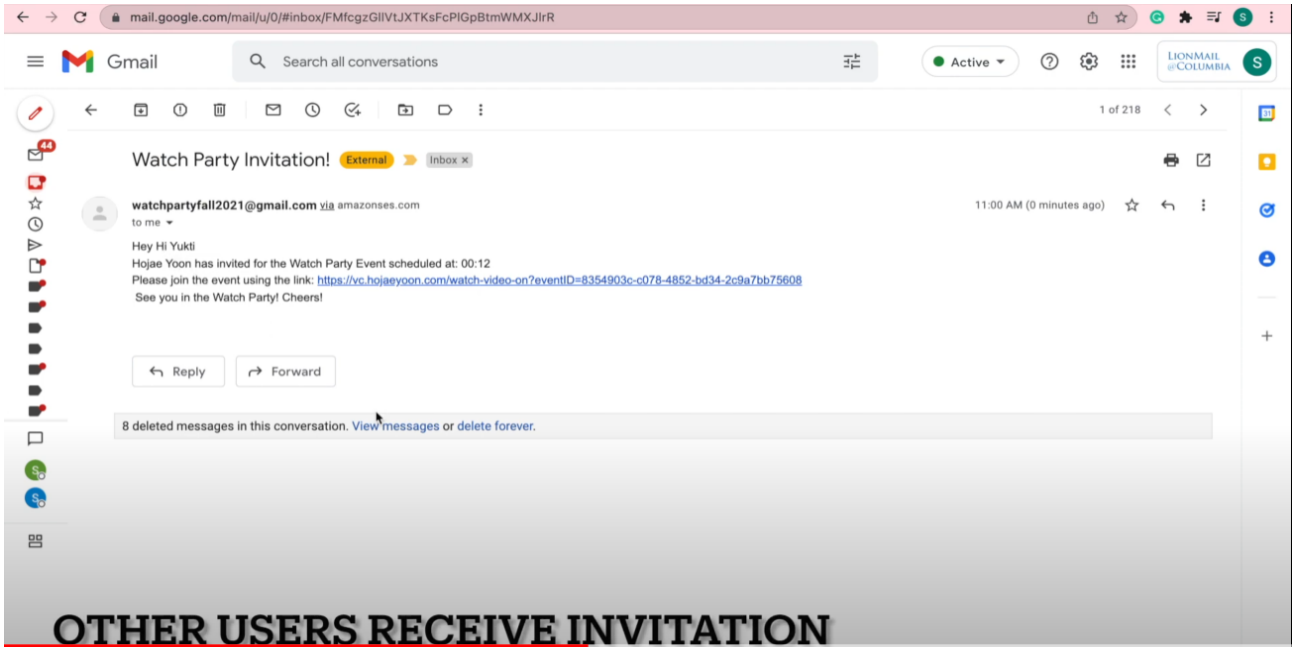


Streaming Experience -

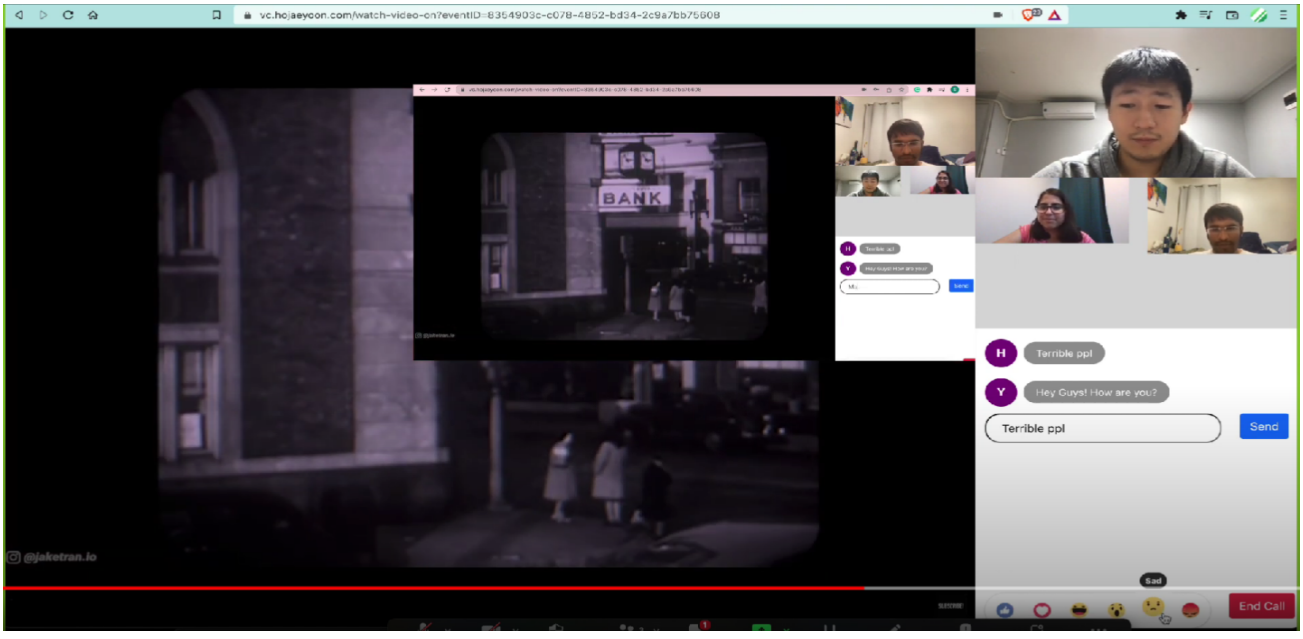
- When users click on the Join Stream button on the dashboard or the invitation link that they receive in their email, they are redirected to the Streaming Page of our application where the video/movie is streamed in sync with respect to each user.

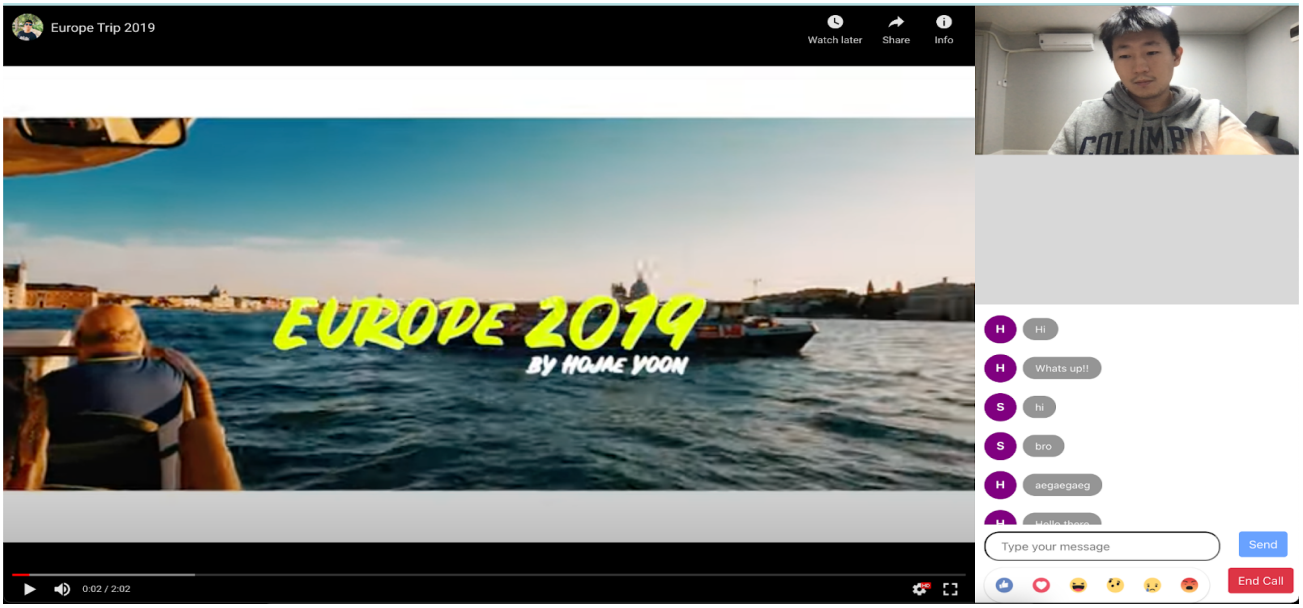


- In case an invited user does not have an account associated with our application or is not logged in, they are automatically taken to the Welcome Page where they can either create an account if they are a new user or log in using their credentials.



- Once logged in, only then the users are allowed to attend the watch party they have been invited to so that no unauthorized user can access the content.
- On joining the stream, users can start watching the video together which is completely synced for each user attending the session. This means that if any one of the user pauses the video, the video is paused for everyone in the video room.
- Similarly, if any user skips forward to a particular timeframe of the video, the same will be reflected for each user. This way all the users are able to have full control on what is being viewed.

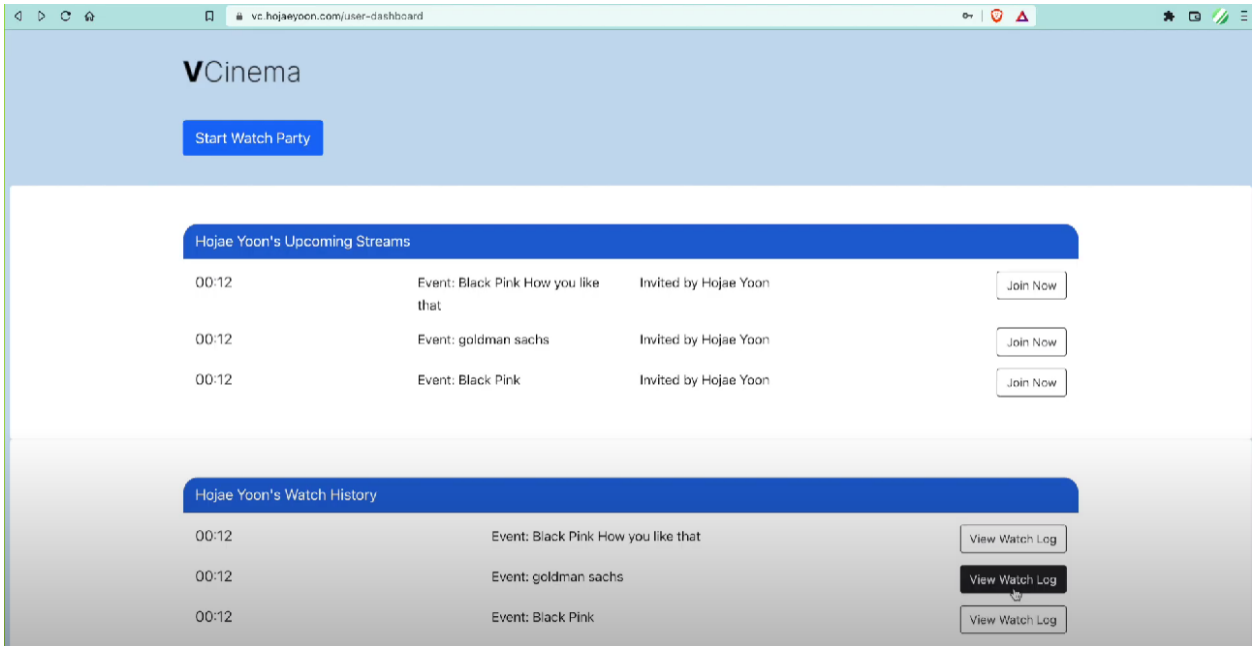




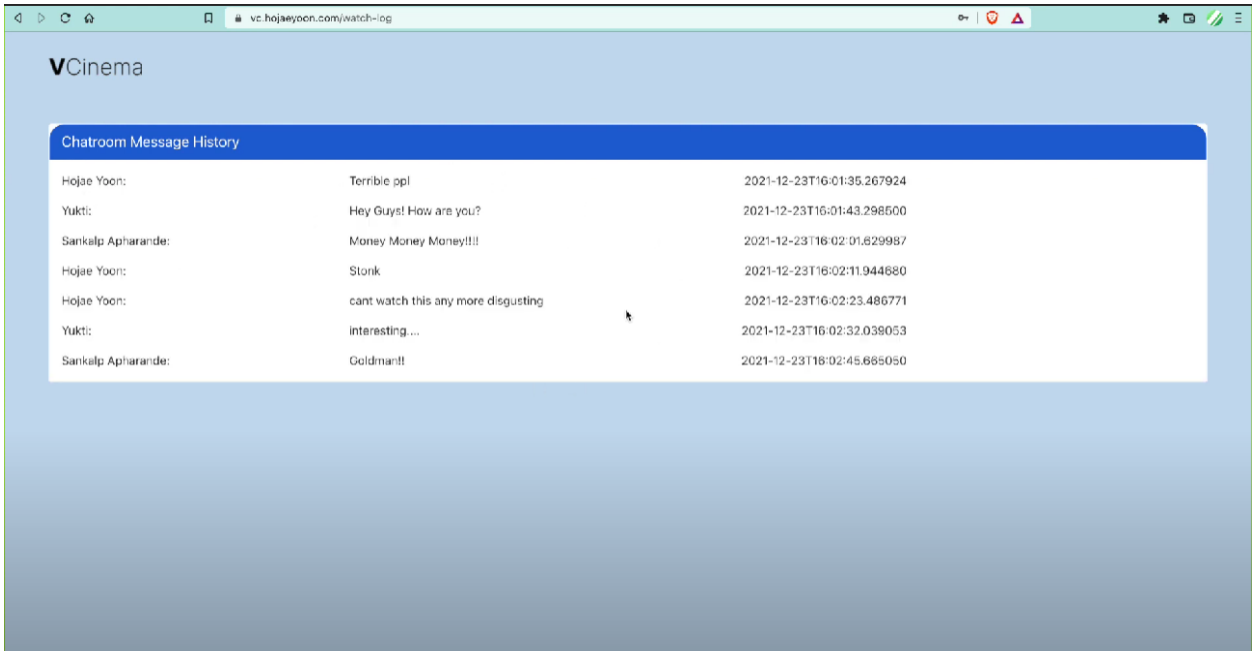
Video Chatting - Our application also supports video chat during the streaming experience to make it even more fun for the user. This means that users can not only watch the video/movie in sync but at the same time see each other live and can have conversations. They can discuss whether they like or dislike certain parts of the video they are watching together which makes the whole experience a lot more interactive. This can be especially helpful if students decide to use our platform to watch lectures or educational material and they can easily hold discussions on various aspects which are not provided by any current commercial platform.

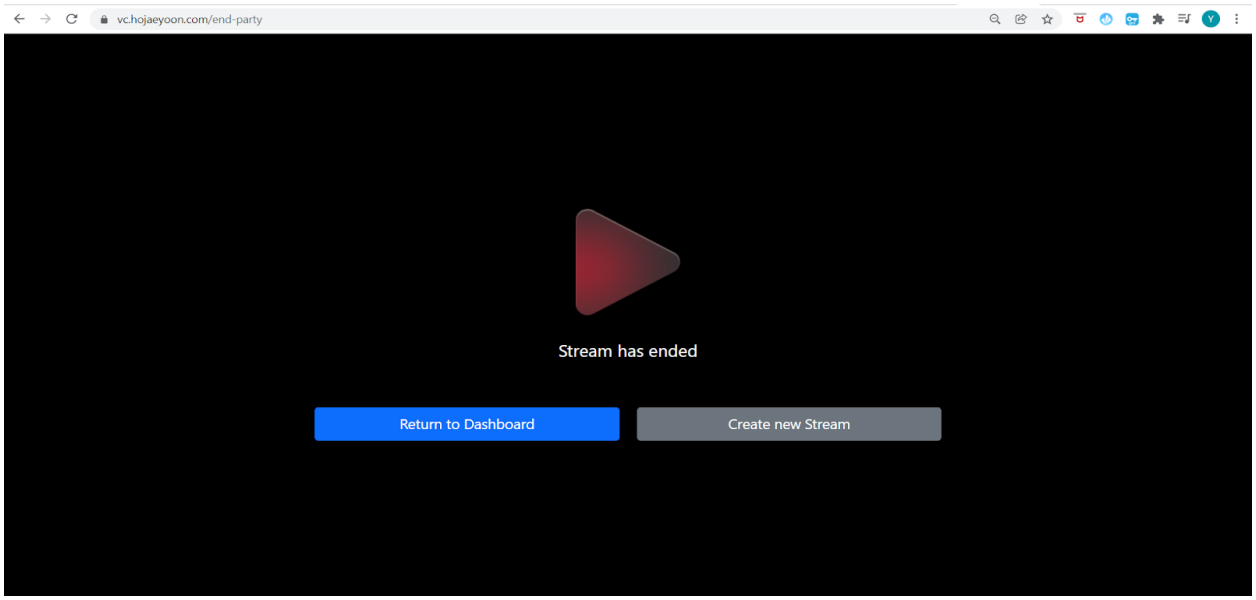
Messaging - Besides Video chatting, users can also send messages which will be displayed on the right top panel to everyone present in the video room during streaming. To make the experience overall interactive, this is a great feature as users might not want to talk when the video is playing but still should be able to express how they feel about various parts of the video. All the messages are synced in real-time, which means that a user message to broadcast to all users present in the meeting.

Watch History - This feature lets users view all the previous streams that users have had on our platform. They can take a deep dive into the memory lane and experience again all the fun they had using our application.



Message History - Another awesome feature of our application is that users can view their message history from various streams. All the fun and important conversations through chat messages that happened during any streaming experience are stored and can be viewed using the “View Watch Log” button in the dashboard.





Users can end the stream at any time they want and can easily rejoin the same stream later on if they wish to do so at their convenience. Once finished, they can choose to start a new video party or return to the dashboard.

6. Future Work

The features we are working on to add to our final demo include fetching the current trending videos, implementing Google OAuth for (login/registration). We also plan on integrating video recommendations based on trending media using data from third-party APIs. Another possibility is implementing collaborative filtering-based video recommendations using AWS sagemaker. We also plan on experimenting with other use cases for the core features that we've implemented. Essentially, our project developed a socket synchronized real-time video player, video conferencing, and messaging. With a slight change of UI, we can rechannel the development into other exciting areas such as education.

7. Summary

Overall, based on the interaction with the professor, we found that our idea was well accepted by the professor. We are really interested in the idea and would like to pursue this idea to roll out an actual product. We sincerely appreciate the professor's inputs on the use-cases of our product and we would steer our app in that direction after doing more market research, assessing more use cases, polishing the application, and further in-depth brainstorming.