## Project Summary

(maximum three sentences): (1) the problem you're addressing, (2) what your solution is, (3) what unique approach you're taking in your solution (how it's different from other similar solutions). This summary will be used as a blurb on the project gallery.

Due to the COVID-19 pandemic, sports fans cannot go to the stadium so they feel less excited and can't enjoy such a passionate atmosphere of the stadium. Through Cheers, they can enjoy the sports game with people and feel like they are in the stadium by sending cheering messages, doing mini-game like Surfing the waves and watching it with other people. These kinds of services usually offers only chatting but our service offers archiving hot viewpoints according to users' cheering and various interactions such as moving user's avatar, chatting(emoticon) with others, and playing mini-games motivated by "Surfing the waves(파도타기)."

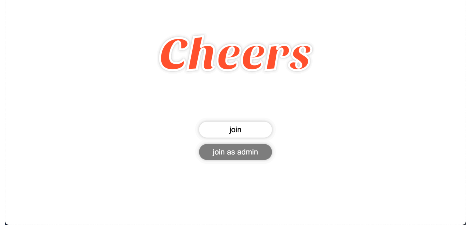
## Instruction

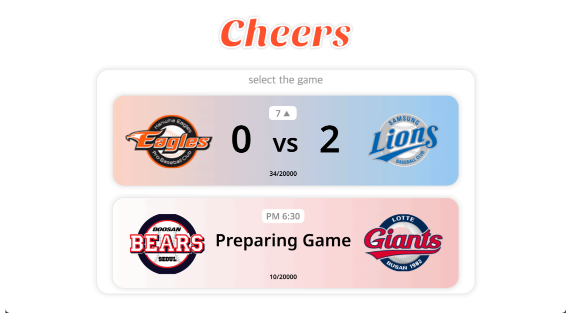
Join Cheers site(http://ssal.sparcs.org:33333/)
Give a quick tour of the interface, and also show off some of the highlights of the interface. Note that this should not cover all features you have; focus on the most exciting and important parts. Use screenshots and callouts.

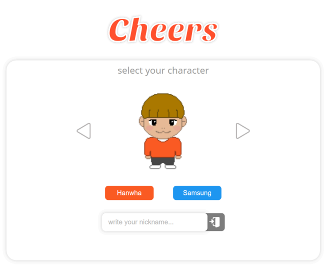
### Task1

[Use characters and interfaces such as emoticon and chatting to help people's interaction]

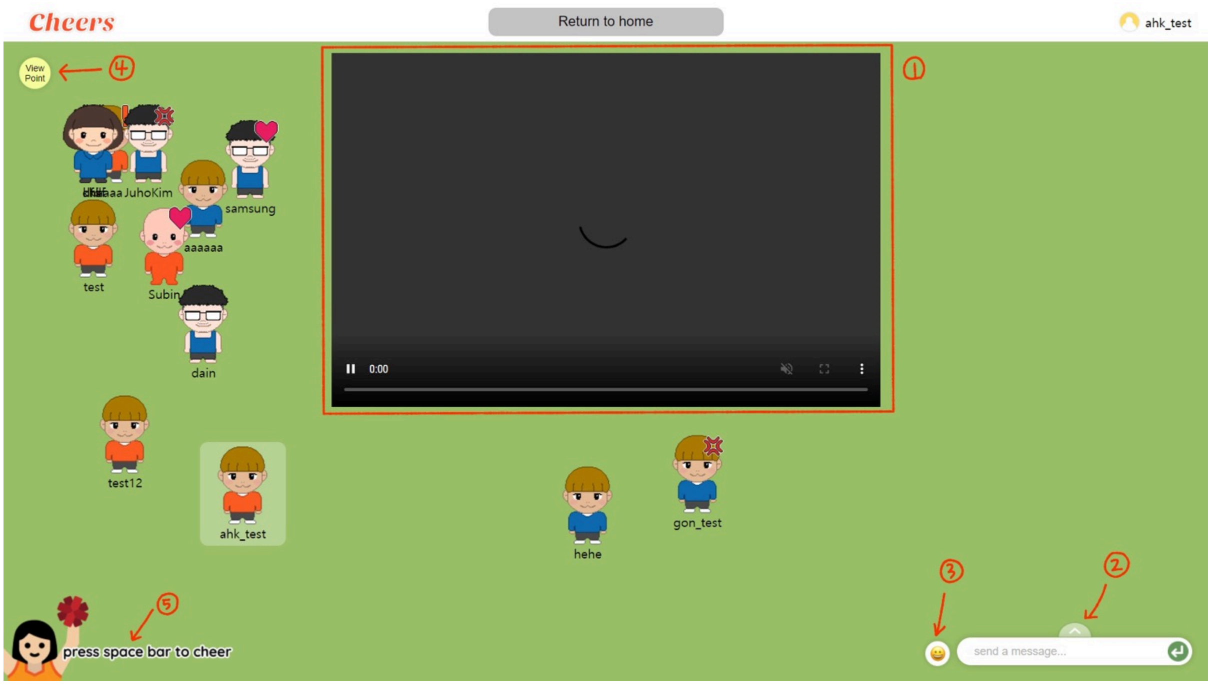
By clicking the "join" button, you can select the sports game which you want to watch.
After selecting the game, select your lovely character and type nickname! The character and nickname represent you!



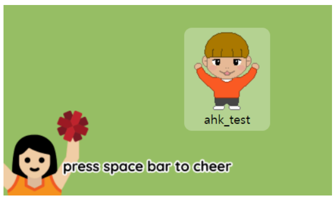




[Picture 0] Game/Character selection page



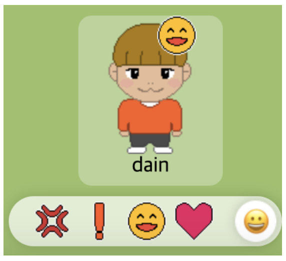
[Picture1] Main Screen



[Picture1-1] Cheering character

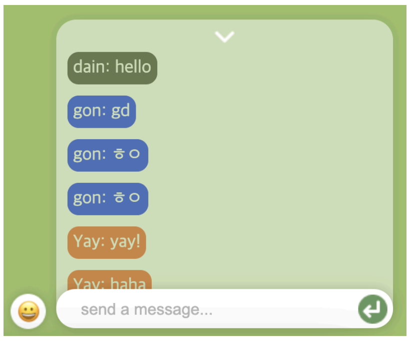
Now, you enter the main page! In this space, you can enjoy many functions such as:

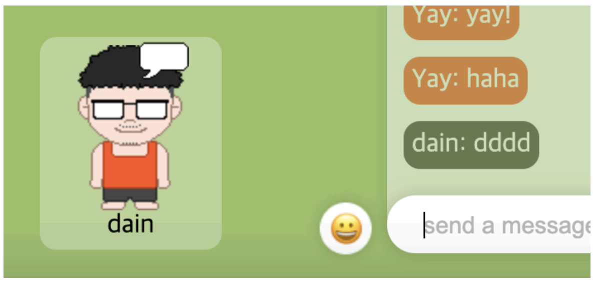
- You can watch sports live stream(①)
- You can move your character using the arrow key. But, **you must first click your character first**.
- You can Chat with people(②) and send emoticons(③)
- You can watch archived hot clips by clicking the viewpoint button(④)
- Yout can Cheer your team by pressing the spacebar(⑤). When you press the spacebar, your character turns into [Picture 1-1].



[Picture 1-2] Emoticon

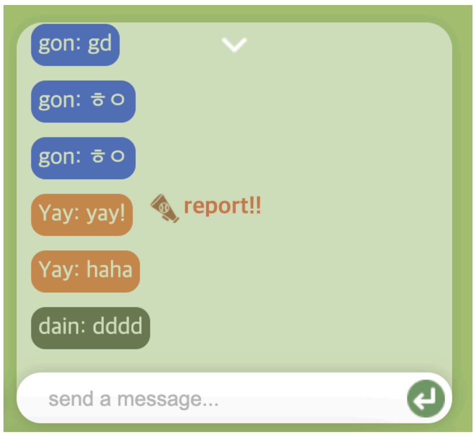
- You can choose the emoticon by clicking the smiley face.
- Your emoticon shows at the left-top of your head
- The emoticon disappears after 5 seconds

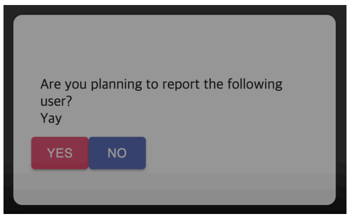

dain: hello
gon: gd
gon: ㅎㅇ
gon: ㅎㅇ
Yay: yay!
Yay: haha


Yay: yay!
Yay: haha
dain: dddd

[Picture 1-3] chatting screen [Picture 1-4] chatting emoticon

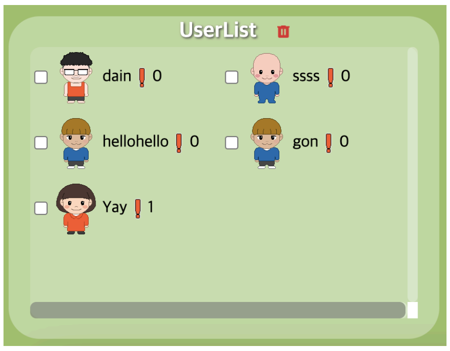
- The chatting shows the other user's chatting.
- If the user who sent the chat is in the blue team, the chat color is blue, and vice versa.
- The chat by you is colored as dark green.
- The user who sent a chat will have a chatting balloon on the top of the head for 5 seconds.


gon: gd
gon: ㅎㅇ
gon: ㅎㅇ
Yay: yay!
Yay: haha
dain: dddd


Are you planning to report the following user?
Yay

[Picture 1-5] Report function [Picture 1-6] Report popup

- You can also report certain users by clicking the chatting balloon.
- After clicking the balloon, if you click the 'report!!' letter, pop up will ask if you will report the user. If you click yes, the user will be reported.


UserList

dain ! 0	ssss ! 0
hellohello ! 0	gon ! 0
Yay ! 1	

[Picture 1-7]

- You can check the number of reports of each user if you are an administrator. [Picture 1-6] is in the admin page. The number beside the '!' is the reported number

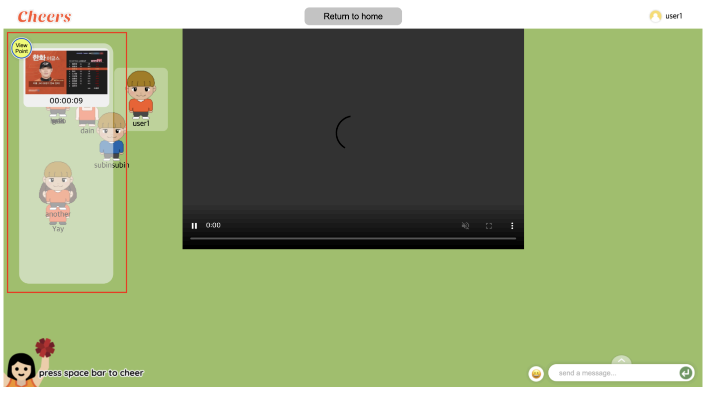
Possible bugs:

- After you join, your character may not show immediately, or may have to refresh the website.
- If you join as multiple users in one computer, some of the accounts may turn into the most current one. It is the problem of using browser cookies in react-redux. **So we recommend you to use the secret window of Chrome.**
- **You have to click your character before you move**
- Past chattings are not saved. Only the chatting that was typed after you joined the site will be shown.
- The position and speed of the characters can differ, depending on the resolution of the user's monitor.
- You should press the Return Home button if you want to log out. However, just by going to the previous page or closing the browser doesn't log you out. By logging out means being erased from the database and not being shown in other user's screens.

### Task2

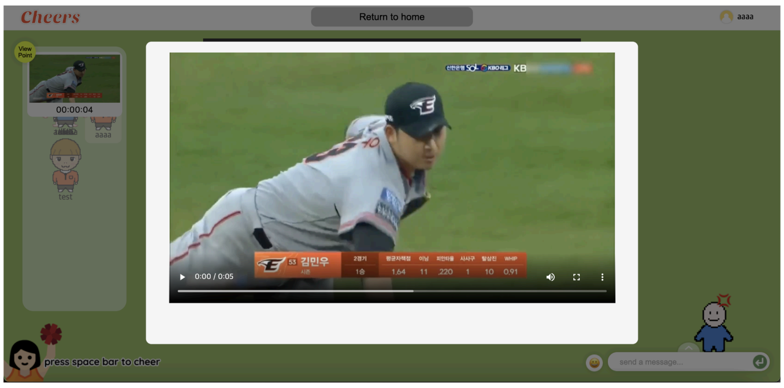
[Archiving important moments during the livestream game based on the number of people's response and sharing the moments with other people]

If the number of cheering times is over some standards, the moment of streaming is archived and it is stored at the viewpoint! In this case, you can see the moment again by using a viewpoint.



[Picture 2-1] Viewpoint dropdown

- You can live stream a video that you want using the OBS software (check the Appendix)
- When the user presses the spacebar, it will count as one cheer.
- If 10 cheers are accumulated in 20 seconds, that moment of video(game video) will be automatically archived.
- You can check it by clicking the viewpoint button on the left, top side of the screen.
- If you click a specific moment in viewpoint menu, video of 10 secs will be played.



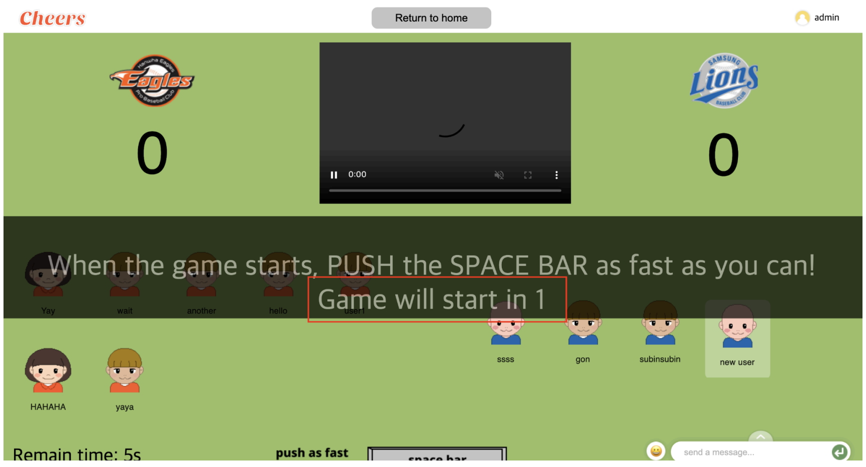
[Picture 2-2] hot clip

Possible bugs:

- When you click the viewpoint button, the page might turn white. This is due to the delay of getting information from the server. In this case, please reload the window. Or you can go back to the main page and login again, then click the viewpoint button.
- The sound of the video live stream is muted in the main page. The video in the administrator page works fine. The video live stream in the minigame also gets muted frequently. We are thinking this is because the sound gets muted every time the page gets rerendered, which is frequently done.

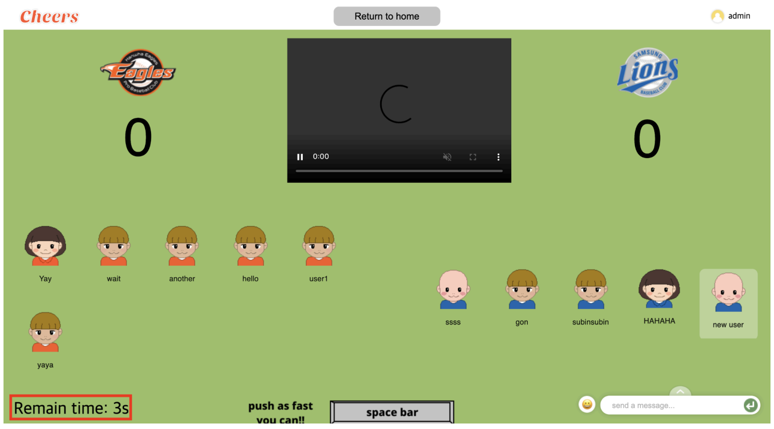
### Task3
[Provide mini events that people can play by groups of teams that can encourage competition]

#### MINIGAME1
[Cheer like crazy]



[Picture 3-1] Before Mini game start, countdown

- When the administrator starts the game, all the user's pages turn into the minigame screen.
- Automatic countdown will be started, and short instruction will be given as well.

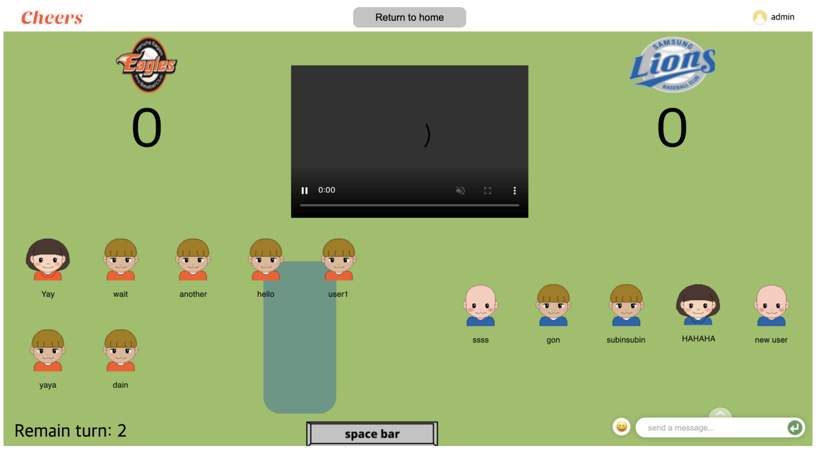


[Picture 3-2] Minigame 1 (**you should click your character first to cheer**)

- The required action to win is to press the spacebar, and every time you press the spacebar, the score of your team will go up.
- The remaining time is shown at the left bottom side.
- When time is out, a team with a higher score will win.
- Considering fairness, you'll get proportional points with your enemy team member's number.

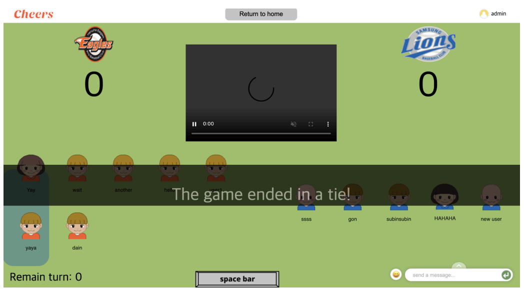
#### MINIGAME2

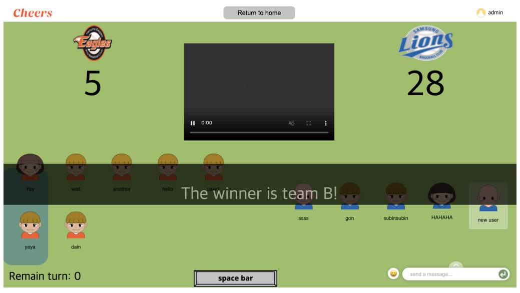
[Surfing the waves]



[Picture 3-3] Minigame 2

- Similar to [Cheer like crazy](Mini Game 1), the timer and instruction will be given.
- Rule of [Surfing the waves] is similar to [Cheer like crazy].
- When the red bar moves to your position, push the space bar and you'll earn a point.
- Scoring policy is the same with Mini Game1(proportional to enemy team member's number). Red bar will pass through for several turns and you can check the remaining turns at the left bottom side of the screen.





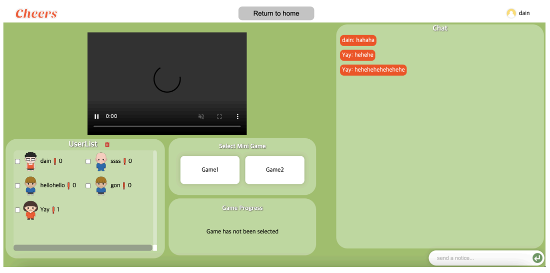
[Picture 3-4] Announce winner after minigame

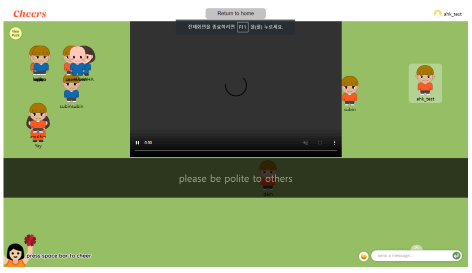
- When the game is over, the system will automatically notice the winner, and you'll be returned to the main screen. Administrator will be returned to the administrator main screen.

Possible bugs:

- **You must click your character before your initial cheer**
- When you press the spacebar continuously, your score can automatically go up in minigame1. Similarly, in minigame2, if you keep on pressing your spacebar when the wave passes, the score will go up.

### administrator





[Picture 4-1] Administrator UI [Picture 4-2] Administrator notice to users

If you are an administrator, you can join an administrator page through "join as admin" in the initial page and type **'hello'**. In this page, you can

- Send a notice message
    1. Administrator's chat message will be announced for every user as in the picture. The notice message will automatically disappear after 4 secs.
    2. Administrator's chatting box is relatively big to make it easy to monitor trolling users.
- Start a mini-game
    3. Admin can start a mini game by clicking the game in the 'Select mini game' section.
    4. If there's too little or too many users to play the game, game start will be restricted. Required number for each team is minimum 3 to maximum 15.
- Kick out some trollers

5. In the Userlist, the reported number is on the left of each user. So administrators can collectively consider trollers with a reported number and chat. Admin can kick out those trollers by checking and clicking a trash can image.

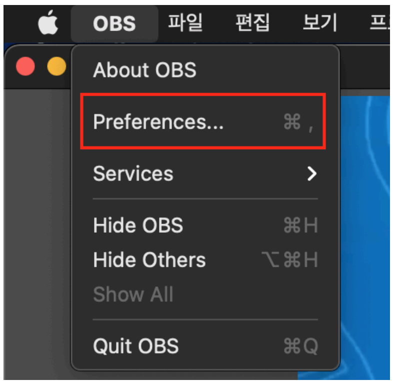
Possible bugs:

- You must type “**hello**” for joining as admin passcode.
- When you logged in as a user, and then returned to home, and joined as admin again, it might not work. We are assuming that this is the problem of browser cookies because redux-persist. In this case, if you reload the page again, join as admin will work.

### How to send streaming video & archive video

You can send your streaming view by using the OBS program.

1. If you don’t have an OBS program, install OBS. (https://obsproject.com/)
2. Execute the OBS program, and go to a Preference

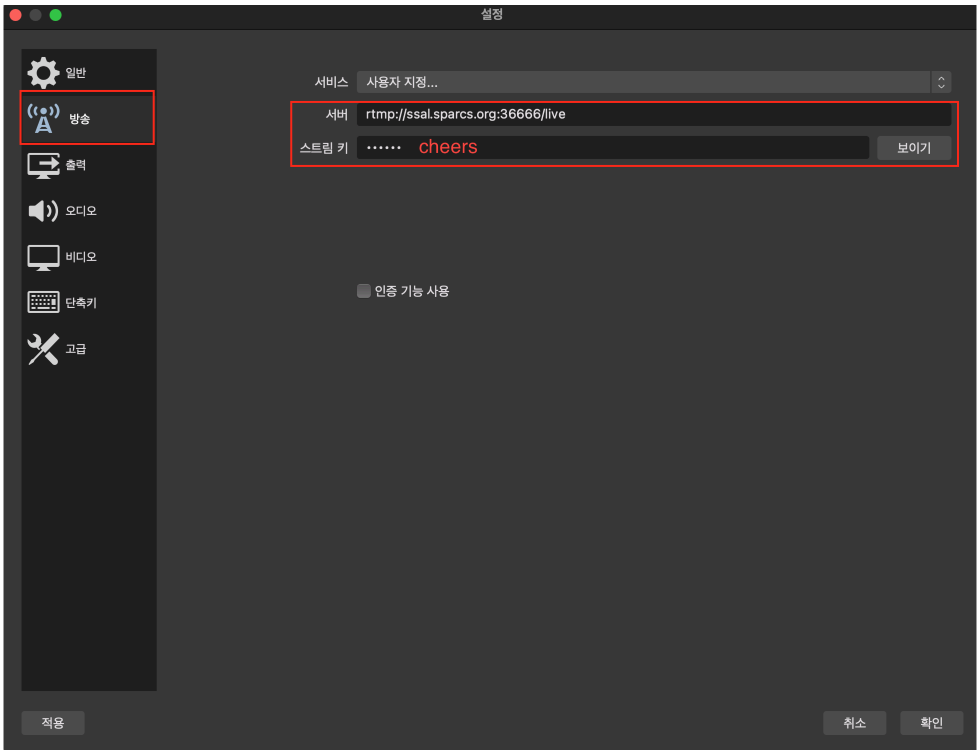


[5-1] Streaming Guide 1

3. In Preference, go to the second tap “BroadCast”, and type server and stream key.

   server: rtmp://ssal.sparcs.org:36666/live

   stream key: cheers



서비스 사용자 지정...

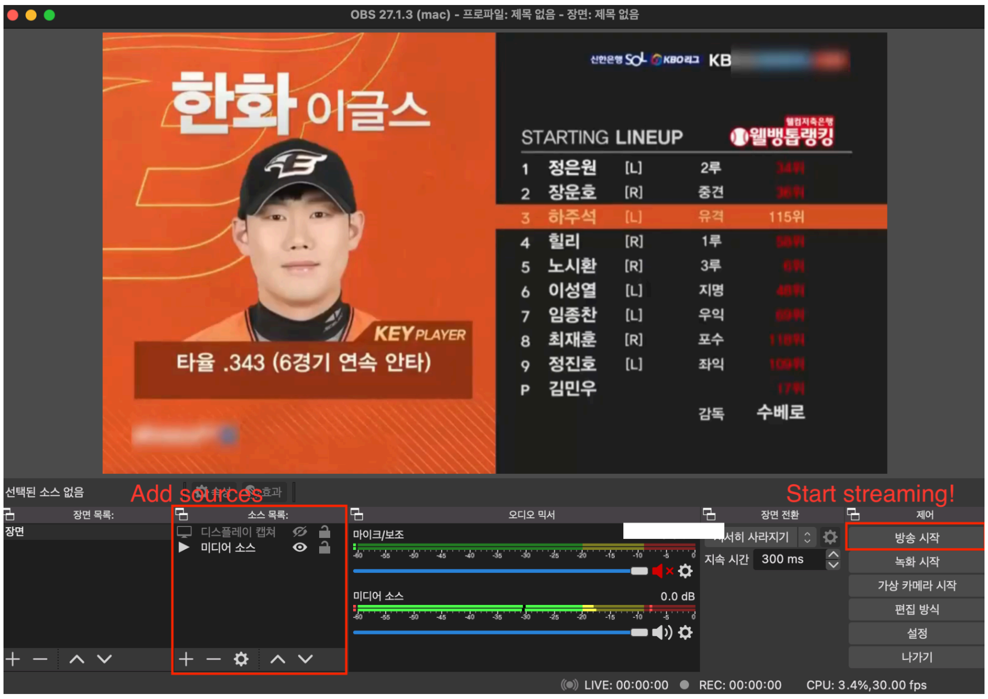
서버 rtmp://ssal.sparcs.org:36666/live

스트림 키 •••••• cheers 보이기

인증 기능 사용

[5-2] Streaming Guide 2

4. Add sources for streaming and click the streaming start button! Then, you can see your streaming at Cheers main page.



디스플레이 캡처

미디어 소스

마이크/보조

미디어 소스 0.0 dB

서서히 사라지기

지속 시간 300 ms

LIVE: 00:00:00 REC: 00:00:00 CPU: 3.4%,30.00 fps

[5-3] Streaming Guide 3

5. To transform and archive streaming video, you need ffmpeg. Try to install ffmpeg.

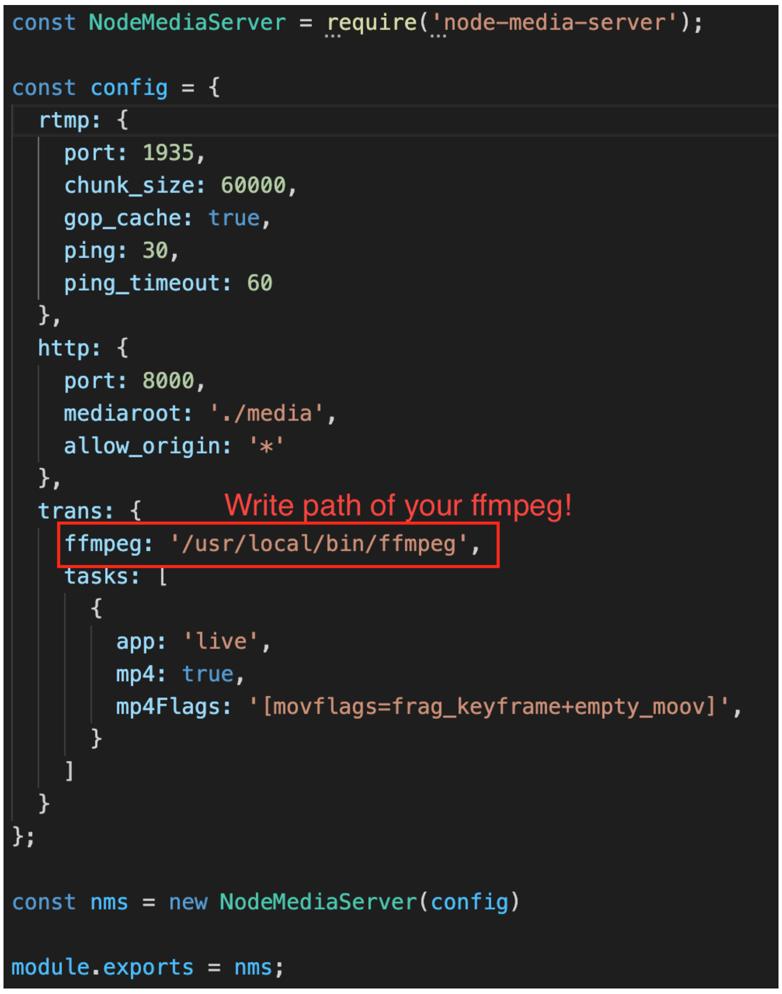
(official: https://www.ffmpeg.org/download.html, ubuntu: `sudo apt update` `sudo apt install ffmpeg

mac: `brew install ffmpeg`)

After installing ffmpeg, check whether the installation is success and the path of ffmpeg.

6. Go to the backend file: `cheers-backend/streaming/index.js`

   Then, change the ffmpeg path into your path of ffmpeg.



```
const NodeMediaServer = require('node-media-server');

const config = {
  rtmp: {
    port: 1935,
    chunk_size: 60000,
    gop_cache: true,
    ping: 30,
    ping_timeout: 60
  },
  http: {
    port: 8000,
    mediaroot: './media',
    allow_origin: '*'
  },
  trans: {
    ffmpeg: '/usr/local/bin/ffmpeg',
    tasks: [
      {
        app: 'live',
        mp4: true,
        mp4Flags: '[movflags=frag_keyframe+empty_moov]',
      }
    ]
  }
};

const nms = new NodeMediaServer(config)

module.exports = nms;
```

[5-4] Archiving Guide

##URL of your prototype

A live version of the prototype for evaluation. Note: the URL must work at least until your assignment is graded. Course staff will run your prototype to do a heuristic evaluation for grading. If the link doesn't work, your team will be penalized. If there are specific requirements (e.g., browser or device settings), include them as well.

Come and enjoy Cheers!

http://ssal.sparcs.org:33333/

##URL of your Git repository

Make sure to add a README file that briefly describes the code, e.g., main JavaScript file, or where main feature implementations are, etc. Several lines are enough.

We have two git repository: Frontend and Backend

Frontend: https://github.com/RyuChigon/cheers-frontend

Backend: https://github.com/RyuChigon/cheers-backend

## Libraries and frameworks

List any external dependencies you used for your implementation (e.g., Bootstrap, Semantic UI, etc.).

- Frontend
    - Main Framework: React Javascript
    - Main Css: Styled-components
    - Sub Css: Material-ui
    - Redux
    - Axios
    - Socket.io
    - EsLint, Prettier, Husky, Lint-staged and Craco (For cooperation tool)
- Backend
    - Main Framework: Node Express
    - Cors
    - Ffmpeg
    - Mongoose
    - Node-media-server
    - Socket.io
    - Nodemon
- Streaming Program: OBS

## Individual Reflections

Each member should write this part on their own, reflecting on their own experience. Merge all members' mini-reports in the final report. Answer the following questions:

- Which part of the system did you directly contribute to?
- What were some of the difficulties you faced?
- List one useful skill you learned while working on the high-fi prototype.

[Chigon]

1. Which part of the system did you directly contribute to?
    a. Cooperation setting
        i. In the cooperation setting, I made git repositories for frontend and backend and invited our team members. We already fixed the main frameworks: React Javascript for frontend and Node Express for backend. So I set these frameworks at the repositories and installed necessary tools like ‘styled-components’, ‘craco’ in advance. In addition, I made a draft (Initial, GameSelect, CharacterSelect, Main Page) of the prototype for setting format. Through the draft, we could divide tasks efficiently and accelerate our service. In the beginning, our team members have different code styles so I installed EsLint, Prettier and Husky for consistency of code and cooperations. These actions offer us guidelines for code forms and smooth cooperations.
    b. Live Streaming and Archiving (Main Task)
        i. I made the streaming by using the OBS program and node-media-server module. Archiving service is made by using ffmpeg program which is a video edit program. On the Main Page, users can see the streaming when the admin opens the streaming services. And when the number of cheering is over the specific counts, the moment is stored at the Viewpoint which is located on the left side of the Main Page.
    c. Character Movement
        i. On the Main page, users can move their characters on the screen. We have 4 lovely characters and I make Character components and if they get specific key values, their position is changed depending on the key values(e.g. Left input -> move left).
    d. Character Emoticon
        i. On the Main Page, users can use emoticons for their expression. The emoticon button is expanded when users click it. The expansion space has four emoticons: angry, exclamation, smile and heart. Also, it disappears after specific times (I define it as 5 seconds).
2. What were some of the difficulties you faced?
    a. The motive of the cooperation setting is immediately after other team members push contents. Their code styles are very different from each other. I worried about bad readability and the bigger the amount of code is, the worse the problem becomes. In the cooperations, readability of code is very important because it should be easy to interpret by other team members for

efficient progress. So I think this situation should be prevented earlier. The solution is the code format tool: EsLint, Prettier. But I wanted a stronger system for formatting and then I found Husky. When code writers do “git commit”, Husky confirms whether the code is well formed by EsLint and Prettier. If it isn’t, it prevents “git commit.” Expectably, these systems prevent horrible code inconsistency and improve our code readability. However, some team members feel hard to use these systems so I only adjusted them in the frontend because the frontend has more codes than backend.

b. The task of streaming and archiving is very unfamiliar with me because I have not done it. I don’t know what specific programs and modules are needed for them. So I spent lots of time searching for streaming and archiving. I thought of a streaming program for streaming and a server for sending it. After lots of searching, I found an OBS program and node-media-server module. I could send the streaming by setting the server and streaming key in OBS. And by using the node-media server module, it could catch the streaming and translate it into video or archive. The streaming service is playing but it is muted. I want to play unmuted but there is a policy of autoplay in Chrome(https://developer.chrome.com/blog/autoplay/) so it is hard to autoplay unmuted. Ffmpeg is a video edit program so it makes the streaming video into cutted video. I use this convert function of ffmpeg for archiving hot viewpoints. First, if users cheer by pressing ‘Space Bar,’ the server get this signal and counts the amount of cheering during a specific time (I define it as 20 seconds). If the amount of cheering is over the standard number(I define it as 10 for the test). ffmpeg makes the streaming video into a 10 seconds short video from the moment and thumbnail file of the short video. And the server sends these archiving data(short video, thumbnail, and progress time) to the frontend and then, the frontend can show the archiving viewpoint list to users. After completing this task, this process seems not difficult, but I had no idea about streaming in advance. So figuring out how to develop these services is very difficult.

3. List one useful skill you learned while working on the high-fi prototype.
    a. I learned about how to operate a streaming service. I wanted to know how the streaming operates because I’m interested in it and I enjoy the streaming services like Youtube and Twitch. In the high-fi prototype, I had to make a sports streaming service so I searched how to develop streaming. As a result, there are many streaming programs to help streamers broadcast. To broadcast streaming, we need a streaming server, stream key to identify it and a player for streaming. It is a useful experience because I had not developed it and I’m interested in streaming services!

[Dain]

1. Which part of the system did you directly contribute to?
    a. Administrator page UI
        i. I constructed an admin page UI(including locating video, user list, game selection section, chatting window, game progress section). And I made a userlist to monitor trollers and made them kicked out when the admin checked users and clicked the trashcan icon. To be specific, in the kick-out function, I made a selected user turn into the initial page and other team members added deleting that user's information in the database.
        ii. I made it possible by clicking the minigame1 or minigame2 button, start mini-game. And made administrator can also move into the game page as well as other users. However, the administrator is not allowed to participate in the game.
    b. Common contribution for minigame1&2
        i. For minigame1, other team members constructed UI and functionality that gained 1 score when pushing the spacebar. I constructed a minigame2 UI and adapted game 1's character component. Both of minigame1&2, I added an automatic countdown before starting the game/during the game/after the game. For before&during the game, the countdown number was added explicitly to the UI. And after the game ends, during announcing the game-winner, a timer was adapted implicitly to make all users back to the main page a few seconds later. Add to this, I made the administrator return to the administrator page, not the user's main page. Except for admin, every other user is made to get back to the user's main page. I implemented resetting the game score as well.
        ii. Also, I made users can't move(cheer) before&after the gameplay. And I changed the scoring policy. Before I changed, one user's one spacebar push would gain 1 point. But considering fairness, I changed it to gain score as the opposing team's headcount. For example, if team A has 3 members and team B has 5 members, team A's team members will get 5 points for each spacebar push. And team B's team members will get 3 points for each push. So it can be balanced although different headcounts between two teams.
    c. Minigame2
        i. I made the moving bar regularly move across the users several times. To implement this, I used the useRef function and setInterval function to make the timer work. And I made users get points when they pushed the spacebar when they're on the moving bar. I had some challenges with this task, so I'm going to introduce two approaches I made.
            1. First, I was planning to calculate the position difference between the user and moving bar by absolute time difference. For example, pretend the game was started at 1 PM 0sec, and if the moving bar moves for 10px/sec, if the user located in (20px, Ypx) pushed the spacebar at 1 PM 2sec, it will count as

a valid score. But this has several problems such as socket communication delay due to various users, or the position(of bar) error due to the various resolutions comes from the difference in computer spec of each user.

2. So I decided to calculate the position difference by utilizing each user's bar's position separately. When the game start signal is transmitted by socket, make the bar move automatically & periodically by the frontend program. So the frontend program can get position difference between bar and user at the moment that user pushed spacebar.

d. Etc.
   - i. I manipulated video size separately for the main page/game page, admin main page.
   - ii. I made the character's position limit to only their window's size when moving on the main page. But according to the resolution difference between users, the user has high resolution can slightly go over the limit.
   - iii. On the game page, I made it to indicate the user's character, especially by adding some background only to his/her character.
   - iv. We had a problem with playing audio in video. Due to the security policy of browsers, the automatically playing video should be unmuted just after rendered. In the main page, it is necessary to continuously re-render due to the real-time communication. So the video was kept unmuted because it's continuously re-rendered. So I figured out this problem by re-ordering the prior of components.

2. What were some of the difficulties you faced?
   a. The most difficult part was implementing mini game2. To be specific, there were three main difficulties.
      - i. In both mini-games 1&2, I used the setInterval function directly to implement the timer. But it causes infinite re-rendering due to the characteristic of reactDOM which is re-render every time the state has changed. So I figured it out by using the useRef function, which doesn't require re-rendering every time. And I also used useRef for every timer-needed function such as countdown before the game starts, remaining playtime, and message of announcing the winner.
      - ii. Second, it was challenging to decide the scoring logic of mini game2. At first, I tried to compare the game start time and the moment that the user pushed the space bar and calculate the time difference to the position difference between the user and the moving bar. But it can't work due to the various network delay over each user. So I decided to utilize the moving bar's position information and user's information. The logic is as follows. The moving bar will move by receiving a start signal from the administrator by socket communication. And every time the bar moves its position will be updated. Also, each users' position exists. When the user pushes a space bar, it will be counted to score if the user's position and the position of the moving bar at that moment are relatively close(around 10px). So I can resolve the

        synchronize & timing problem between several users' real-time actions.
    iii. Third, the mini-game scoring policy is inherently affected by the number of each team member. So I should design it fairly with a different headcount. So I made a rule that gains a score as a headcount of the opposing team. So it could be balanced.
  b. Add to this, frankly, building an application that offers real-time communication and mini-game was especially hard for me since it was my very first time developing a web application(javascript, CSS, react, everything). But I overcame the difficulty by focusing on and absorbed in the work. I'm proud of myself. I want to tell me that I did a great job :)

3. List one useful skill you learned while working on the high-fi prototype.
   a. The most useful skill was collectively utilizing the useRef function, socket communication for real-time communication(in our case, especially for the mini-game). Basically, react components are re-rendered when the state has changed. So every variable is initialized and every logic of functions is replayed. So some time-dependent functions like setInterval() are dependent on common state variables, it has a critical danger of an infinite loop. So I used the useRef function rather than state, to prevent re-rendering every time. It was especially effective to implement a moving bar in minigame2. The first time that I implemented it with state, the bar was moved like crazy(infinitely re-rendered…it was really ridiculous). By using useRef functions, I could implement not only a moving bar but also setting the game state as before/ongoing/after.
      In conclusion, to implement real-time communication management with the sequential process, it is very useful to communicate with socket, manage state with the useRef/useState function appropriately.

[Subin]

1. Which part of the system did you directly contribute to?
    a. **Initialized & Managed backend database**
        i. Used **MongoDB atlas** as database
        ii. **Connected MongoDB and server** with mongoose and express
        iii. **Connected Server and Frontend** using axios, and utilized it as a module.
        iv. I had a hard time overcoming the **CORS policy**. I added headers to solve it.
        v. Established user database structure using **mongoose schema**
        vi. Initialized the react redux containing the information for userlist, current logged in user and the score of each users
        vii. Saved the user information of the logged in user from the character selection page, and saved the information inside the database.
    b. **Real-time communication** of emoticon and movement using database and socket.io and redux
        i. I stored the emoticon and the positions are stored in the database
        ii. Every time a position or emoticon changes, the information is sent using the socket.io. This is also saved in the database
        iii. Then, I made each character receive the response of socket.io including the changed position, emoticon, and the user name, and applied it to the character
    c. Established **synchronization of scores** in minigame 1& 2
        i. I **added the redux state** for keeping scores of each team, and also added a mongoose schema for scores in the database.
        ii. Then, every time the score goes up, I update the database using socket.io and mongoose, and return the scores to all the other users using socket.io. The users updated their own redux score states using the received information from the socket.io
    d. **Minigame 1 & 2 UI** using material ui
        i. I used ImageList in material ui to set the characters in neat positions.
        ii. I iterated all the userlist received from the database, and mapped the list inside the ImageList.
        iii. I also used TableRow and Table Container to lay out the video, scores of the two teams, and the user list. The first row contains the scores of each team and the video, and the second row contains the users laid out by ImageList.
    e. **Minigame2 timer** implementation
        i. After the basic UI of minigame1 was done, minigame2 UI needed to be implemented. However, the movement of the block that

represented the ‘wave’ timing was not working properly, and the teammate who was in charge was having a hard time

   ii. I helped the teammate out by using the codes used for timer function, using useRef and setInterval functions. I tried to find the implementation of stopwatch in react that can enable stop, reset, and start, and implemented the second counts as the position of the wave. This way, the wave would move in synchronization with the time.

f. **Report and Return Home** function in administrator page
   i. I added a report modal that would pop up when you clicked each chatting box.
   ii. If you click yes in the modal, it can report the owner of the chatting box. I added a **report** variable in the database of each user and kept the report counts. If someone reported someone else, I sent this information to the server using socket.io, and saved the information inside the database using mongoose.
   iii. I called back the report information using mongoose from the database and sent the information to the administrator page. Then, I showed the number of reports of each user mapped in the ImageList.
   iv. If you press the Return Home, the user gets deleted from the database, which is also applied in other user’s pages. I implemented this by sending the information of the user who clicked ‘Return Home’ and deleted the user from the database in the server.

2. What were some of the difficulties you faced?
   a. One of the main problems was the **synchronization**. I had to find a way to synchronize all the user’s emoticons and positions. I tried to implement this using socket.io, but this was a problem, because when there were many users, I could not send the huge set of information every time.
   b. I moved the character emoticon and position functions to a separated character component, and changed the emoticon or position setting of each character every time someone changed something. Thus, I only emitted and received only one user’s movement at a time through socket.io.
   c. **Minigame UI** was a big problem. I did not know how I could place all the users in a neat row. At first, I considered using the Table UI in material Ui, then found ImageList UI in material ui which solved the problem. CSS is always harder for me, even more than redux and database.
   d. **Synchronization with the game scores** in minigame1 and minigame2 was also a big problem. The changed scores would not be right in every user, and each user’s scores for the two teams would differ. At first I only used socket.io, but then, I added redux and database to emit the correct scores to every user.
   e. Overall, after the basic settings were set, such as the redux, database, and socket.io, other teammates also seemed to utilize the settings more comfortably. I think setting up the initial modules to be used is one of the hardest parts in developing.
3. List one useful skill you learned while working on the high-fi prototype.
   a. I learned how to use socket.io, which I did not before. Socket.io is very useful in real-time connections.
   b. I was aware of react js, because I had two projects working with react js for a short period of time, previously. However, when I recall the first time I used

mongoDB and react.js (which was in 2020 summer of 2020) I spent numerous hours trying to figure out how to connect the back-end and the front-end, and still ended up crying for help. But in this project, I could connect the server and frontend quite easily, which was a big step forward for me.

c. I also learned the skill of organizing the project, because one of the teammates was very skilled with organizing components and setting modules. I learned the efficiency of organizing code, especially in bigger projects.

[Huikyeong]

1. Which part of the system did you directly contribute to?
    a. Design image sources
        i. Character images: We used images drawn by each member in low-fi prototype, the pixel size does not fit with each other and there is not enough image source for cheering, emotion, and movement, so we had to make a new unified character design. Characters with the same body ratio were drawn with 100px*100px. Basically, it was drawn in the direction of maintaining the identity of the original characters drawn by other team members. However, in terms of character diversity, I added middle-aged male character because I thought there would be many middle-aged male baseball fan. Each character needed a total of 7 versions of the basic front posture, cheering posture when pressing the space bar, two walking posture left and right, the back, the upper body for mini-game, and cheering for the mini-game.There are 2 teams and 4 characters, so I created 56 character image sources.
        ii. Emoticon images: In addition, an emotional expression image that goes well with the character of the dot image was created. Also, since the existing image was not good to see because the pixel was broken, I drew an emotional expression image myself. Image sources were created for four emotional expressions of anger, surprise, heart, and laughter.
        iii. Change image sources: The character object, which was using the existing character as a source, was changed so that all new image sources can be used. Since there was no image of cheering, it was possible to implement a character UI that responds to the user's manipulation for four characters and two teams by modifying all parts that did not include logic to change the image source when the space bar was pressed.
    b. Chat box
        i. chatting system: A chatting system between users was implemented using socket.io.
        ii. chat box UI
            1. chat box: When users press the expand chat box button, our service had to show the messages of users they received through socket in the expand field users can see. Therefore, the messages received were managed by putting them in an array called chatArr, and implemented so that they could be

seen by pressing expand button. In addition, a button that can be closed again when unfolded was needed, so if users press expand button, users can see the notexpand button, and if users press expand button, users can see the object alternately.

2. chat balloon: A chat balloon surrounding the chat message is needed, and different chat balloon colors are implemented for each team. It could be implemented using sender information and team information of messages received from msg-rcv. If the current user's name and sender's name are the same, it is displayed as a black speech balloon, otherwise, if the team is Hanwha, it is displayed as an orange speech balloon, and if it is samsung, it is displayed as a blue speech balloon.

c. Administrator communication
   i. administrator chat box: “msg-rcv” socket used by users allows administrator to receive messages so that administrator can check users' chat contents.
   ii. administrator notice: It is implemented so that all users can receive a message when the administrator enters a message using an “admin-msg-snd” socket and an “admin-msg-rcv” socket so that the administrator can notify the users. Messages from administrator are notified to users in the form of notice boxes. A notice appears in white letters in a translucent black box and is implemented to be seen for 4 seconds.

2. What were some of the difficulties you faced?
   a. Chat box
      i. Socket communication between many people: In the past, when I implemented a chat system using socket.io, it was 1:1 communication, so I thought a lot about how to let many people see the message sent by one person. All users are connected to the socket of the server on the backend, and when one user emits the message to the 'msg-snd' socket, the server receives a message (sender nickname, message content, team information) and all users' 'msg-snd'.
      Through the process of *user(msg-snd emit) -> server(msg-snd on) -> server(msg-rcv emit) -> user(msg-rcv on)*, all users can see messages sent by one user.
      ii. Message duplication problem: When I sent one message, one message returned well, but when I sent the next message, there was a problem that two messages arrived, followed by four messages and eight messages, which kept doubling to output. When checking the console of the backend (server), it was confirmed that one message was received, but it seemed to be output several times when the user accepted messages with the msg-rcv socket. This problem was solved by registering a callback when msg-rcv was performed with useEffect.
      iii. Putting newline in react: I put the message in chatArr and used the map to print out the messages. Strangely, however, the line did not change after one message was printed. Various methods such as '\n' and '\r\n' were attempted, but they were not solved. This is because \n is not

recognized as newline in react. Therefore, the problem could be solved using the </br> tag.

    iv. Auto scrolling: The css style attributes of *overflow: auto; overflow-x: hidden;* has been added to make it closer to the common chatting UI so that users can scroll when chatting accumulates. However, the problem is that the chat is added down, but the scroll is always fixed up, so the users have to look down themselves. Therefore, when the *scrollheight* is greater than 0, the *scrolltop* is changed to scrollweight to implement the automatic scroll function.

  b. Administrator communication

    i. To disappear notice after 4 seconds: When users received admin-msg-rcv, they registered a callback using useEffect. In the beginning, the setTimeout was used to turn on and off the notice box alternately with true and false. However, as a result of implementing it in this way, it repeatedly turned off and on without showing for less than 4 seconds if several notices were sent quickly and in succession. Therefore, when the notice message arrives, the state of the timer variable is changed to timer + 1 to count the number of timers, and when set Timeout is called, the number of timers is reduced to timer => timer - 1. Eventually, when the value of timer was 0, the notice box was not shown. If implemented in this way, you can wait 4 seconds from the last notice.

3. List one useful skill you learned while working on the high-fi prototype.

  a. The most useful skill I learned was socket.io. I used socket.io to implement a chat system between users and an administrator to check users' chats, and an administrator to notify all users. With socket.io, I can send and receive data using emit and on as long as I know the socket of a specific name. I knew it originally, but what I learned this time was that I could implement a number of communication using a server. When implementing a chatting system between users, a process of client (msg-snd emit) -> server (msg-snd on) -> server (msg-rcv emit) -> client (msg-rcv on) was used. This is implemented by separating the sending socket room and the receiving socket room. And the server is responsible for receiving information from one user and transmitting it to all multiple users. It's very simple, but I think it's been a strong help to our service. In addition, although it was not a function I implemented, I felt that it was useful because information such as position could be exchanged faster than using DB.