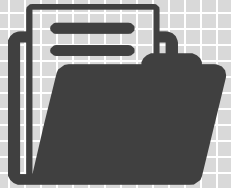
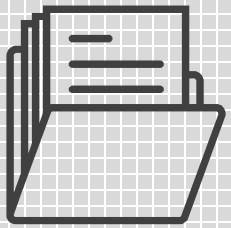
2022 Smilegate Winter :// Dev.Camp



FINAL



Team Ottogi

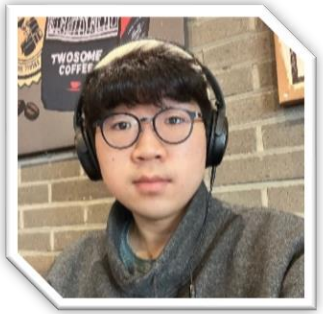
Team Ottogi

최종 발표 자료

FRONT END : 김현우 허다은
BACK END : 김수찬 박규현 백종인

Ottogi 팀 소개

저희 팀원을 소개합니다!



FE / 김현우

FE / 허다은

BE / 김수찬

BE / 박규현

BE / 백종인

2022 Smilegate Winter :// Dev.Camp

주제

Discord Clone Project

Discord ?

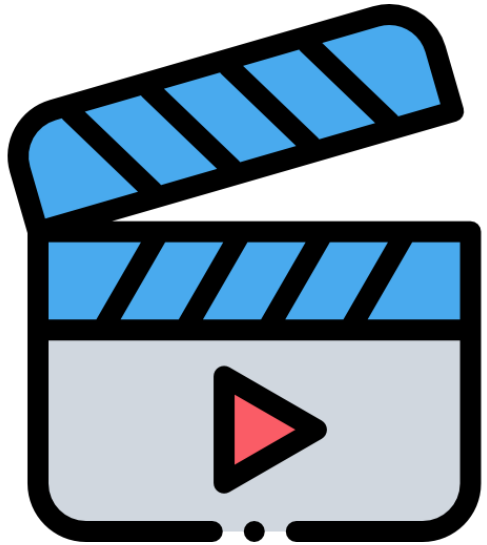
- ✓ 음성, 채팅, 화상통화 등을 지원하는 인스턴트 메신저
- ✓ 대한민국에서는 주로 온라인 게임을 즐기는 사람들이 많이 이용하는 편이며, 게임용 메신저의 대명사.
  - ***실시간 채팅***
  - ***실시간 음성***
  - ***개별 서버 단위의 메신저***

주제 선정 이유

- 팀원들이 평소에도 대부분 사용해본 서비스로 이용 경험이 다수 있다.

- 채팅서버, 시그널링 서버, 알림 서버, 인증 서버, 상태관리 서버 등 기능에 있어 복합적으로 이루어져 있고, 다양한 기능의 구현을 통해 성장을 원하는 팀원들의 니즈에 적합하다.

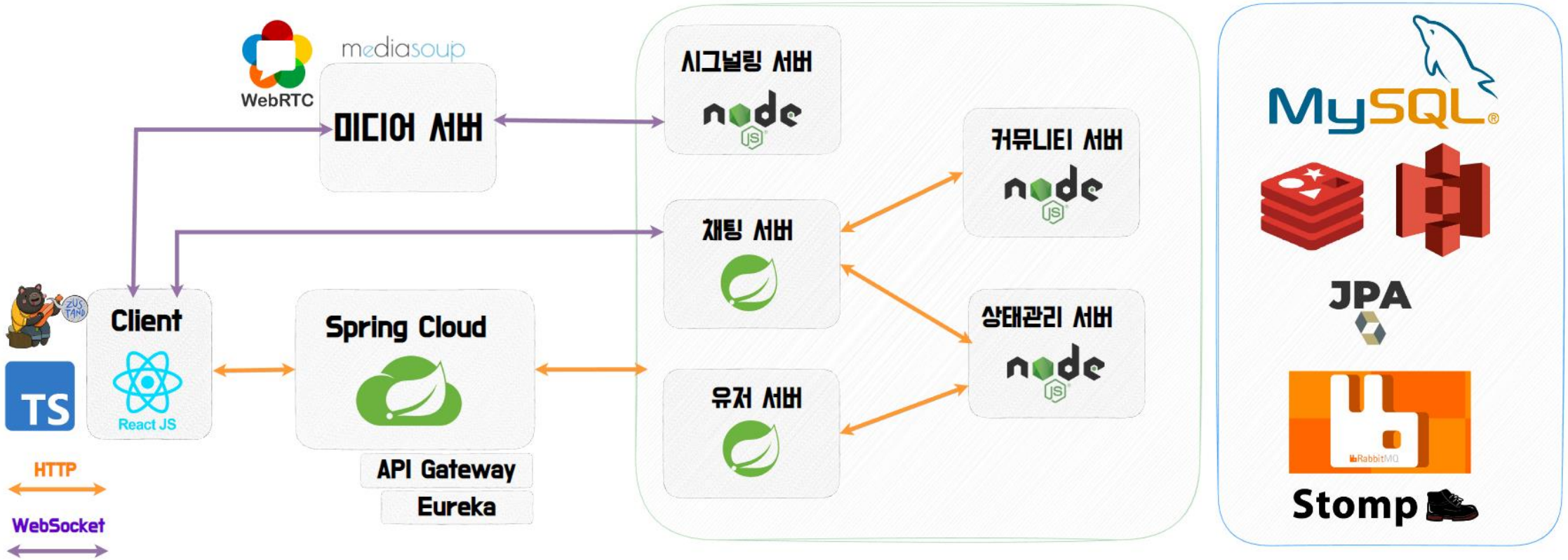
- 채팅, 음성, 화상대화 등 각 기능을 구현 하는데 있어서 팀원들이 기존에 접해보지 못했던 아키텍처와 기술 스택들이기 때문에 도전 목표에 적합하다.



시연 영상

아키텍처

Architecture



업무 분담

Front

김현우, 허다은

UI / UX 설계

디스코드 디자인 시스템 구현

WebSocket 연동

Back

김수찬

WebRTC, Signaling Server

박규현

Community, State Server

백종인

API Gateway,
User, Chat Server

개인 결과물

FE / 김현우

- 서버설정, 사용자 설정 페이지 구현
- 커뮤니티 리스트 구현
- 아토믹 디자인 패턴을 이용한 컴포넌트 구현
- 1대1 채팅, 커뮤니티 채팅 구현
  - FE, BE Websocket 연결

김현우 개인 결과물

WebSocket

실시간 채팅 서버 적용을 위한
채팅서버와 프론트 WebSocket 통신

사용자의 현재 위치 파악을 위해
채널 ID, 사용자 ID 정보를 Header를
통하여 전달

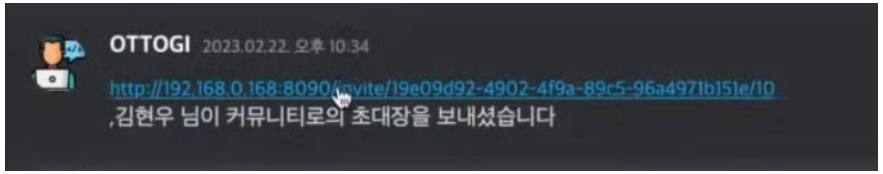
```
const connectChatRoom = () => {
    if (userInfo.id === -1) return;

    client = new Client({
      brokerURL: process.env.REACT_APP_BROKER_URL,
      connectHeaders: {
        channelId,
        userId: userInfo.id.toString(),
      },
      onConnect: () => subscribeChatRoom(),
    });
    client.activate();
  };
```

```
const subscribeChatRoom = () => {
  if (client) {
    client.subscribe(`/topic/${channelId}`, (data) => {
      const { message, name, createdAt, imagePath, type } = JSON.parse(
        data.body
      );

      addChatLog({ message, name, createdAt, imagePath, type });
    });
  }
};

const disconnectChatRoom = () => {
  if (client?.connected) {
    client.deactivate();
  }
};
```

초대 링크 구현

초대 요청 직후에 초대 링크를 채팅으로 보내줄 수 있도록 로직을 구성

채팅에서 초대링크를 클릭해 수락을 할 수 있도록 채팅 메시지를 파싱(parsing)하여 링크로 만들어 전달




```
const onSendInvite = () => {
    sendInvite({
      communityId,
      userId,
      shortUrl,
    });
    sendInviteToChat({
      sender: userInfo.name,
      channelId: channelId,
      linkMessage: `${backUrl}/invite/${shortUrl}/${userId}`,
    });
    console.log(`${backUrl}/invite/${shortUrl}/${userId}`);
  };
```

```
const hasLink = useMemo(() => {
      return /(https?:\/\/[^\s]+)/g.test(text);
    }, [text]);
    const words = text.split(" ");
    const link = words[0];
    words.splice(0, 1);
    const chat2 = words.join(" ");
```

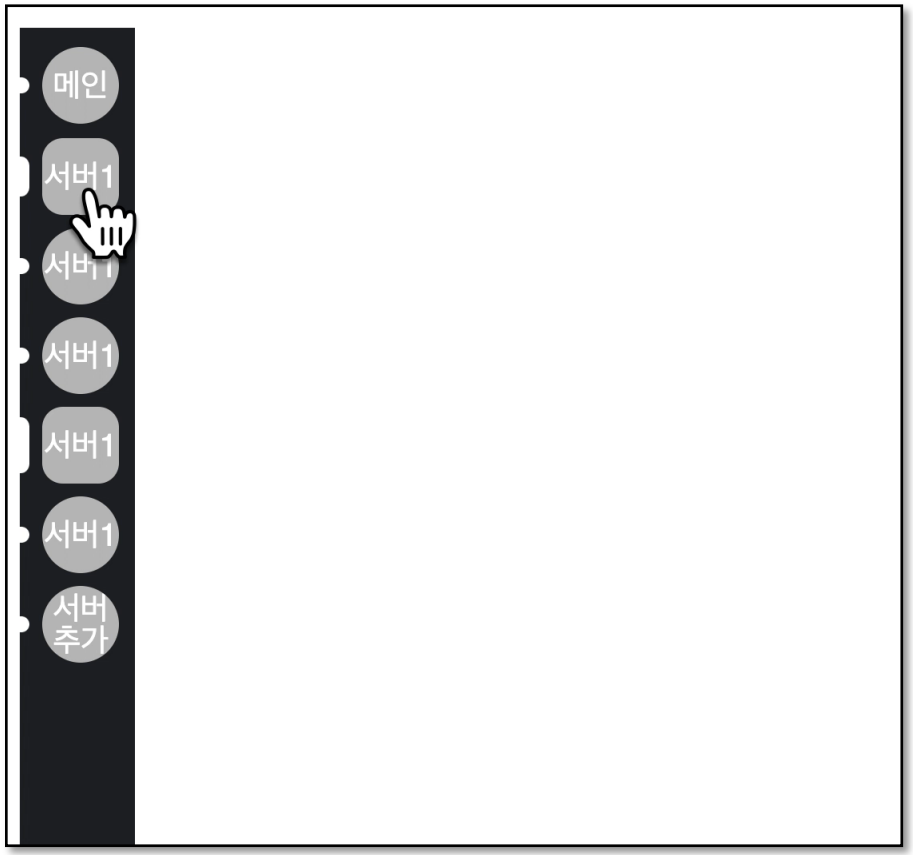
FormData

FormData를 통한 사용자 개인 설정,
서버 설정 시 데이터 전송

React-Query Mutation를 통해
효율적 비동기 적 처리와 데이터 전달

```
let formData = new FormData();
  const [img, setImg] = useState<Blob | undefined>();
  const MakeServer = () => {
    formData.append("communityName", name);
    formData.append("userId", JSON.stringify(userInfo.id));
    if (!img) return 0;
    formData.append("img", img);
    formData.append(
      "profile",
      JSON.stringify({ userName: nickName, img: null, 한줄소개: "한줄소개" })
    );
    createServer({ formData });
    navigate(-1);
  };
```

시행착오 / Drag and Drop



Drag and Drop을 통한 커뮤니티
리스트의 순서 변경
👇
React-beautiful-dnd를 통해
구현을 진행, typescript 적용과
순서 저장 관련 이슈로 미완성

개인목표 - 김현우

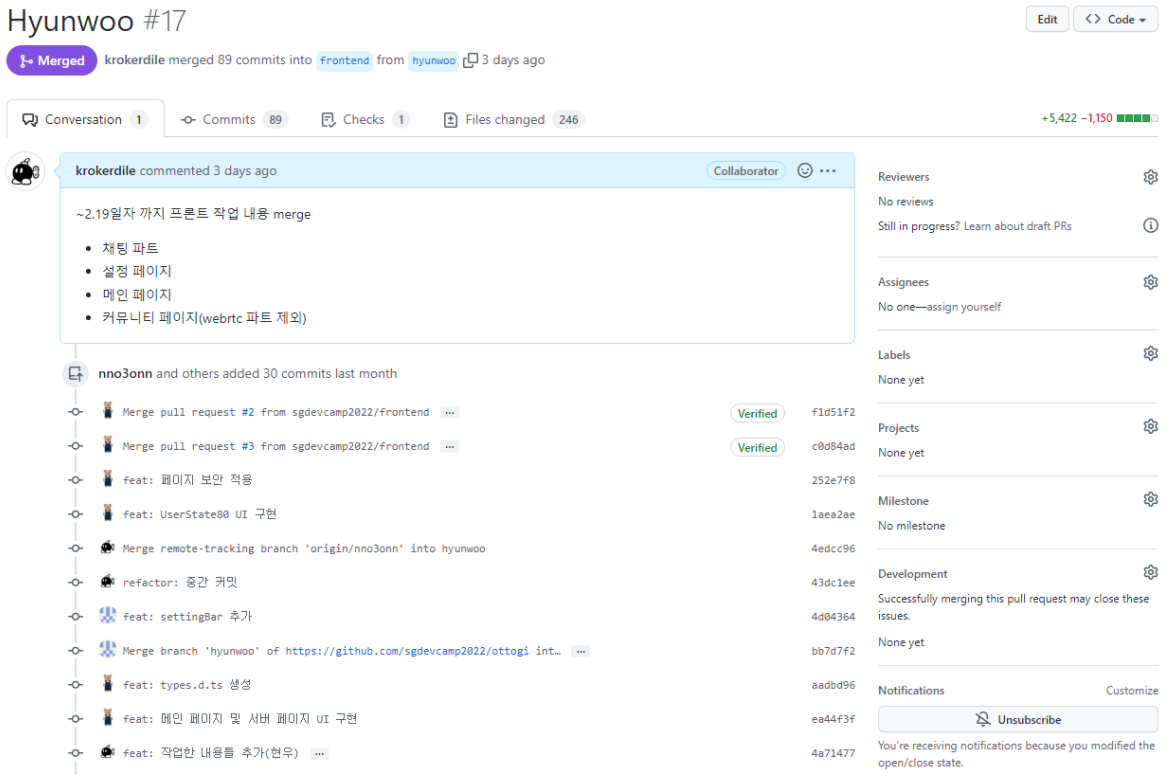
GOAL

GitHub issue, pull request 활용

👇

GitHub Commit 컨벤션 적용, Merge시

Pull Request 통하여 진행



Hyunwoo #17

Edit · Code

Merged krokerdile merged 89 commits into frontend from hyunwoo 3 days ago

Conversation 1 · Commits 89 · Checks 1 · Files changed 246 · +5,422 −1,150

krokerdile commented 3 days ago · Collaborator

~2.19일자 까지 프론트 작업 내용 merge

- 채팅 파트
- 설정 페이지
- 메인 페이지
- 커뮤니티 페이지(webrtc 파트 제외)

nno3onn and others added 30 commits last month

Merge pull request #2 from sgdevcamp2022/frontend	Verified	f1d51f2
Merge pull request #3 from sgdevcamp2022/frontend	Verified	c0d84ad
feat: 페이지 보안 적용		252e7f8
feat: UserState80 UI 구현		1aea2ae
Merge remote-tracking branch 'origin/nno3onn' into hyunwoo		4edcc96
refactor: 중간 커밋		43dc1ee
feat: settingBar 추가		4d04364
Merge branch 'hyunwoo' of https://github.com/sgdevcamp2022/ottogi int…		bb7d7f2
feat: types.d.ts 생성		aadbd96
feat: 메인 페이지 및 서버 페이지 UI 구현		ea44f3f
feat: 작업한 내용를 추가(현우)		4a71477

Reviewers
No reviews
Still in progress? Learn about draft PRs

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
Successfully merging this pull request may close these issues.
None yet

Notifications · Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

개인목표 - 김현우

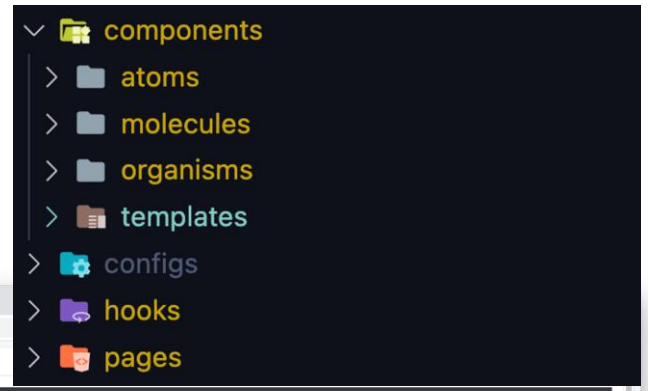
GOAL

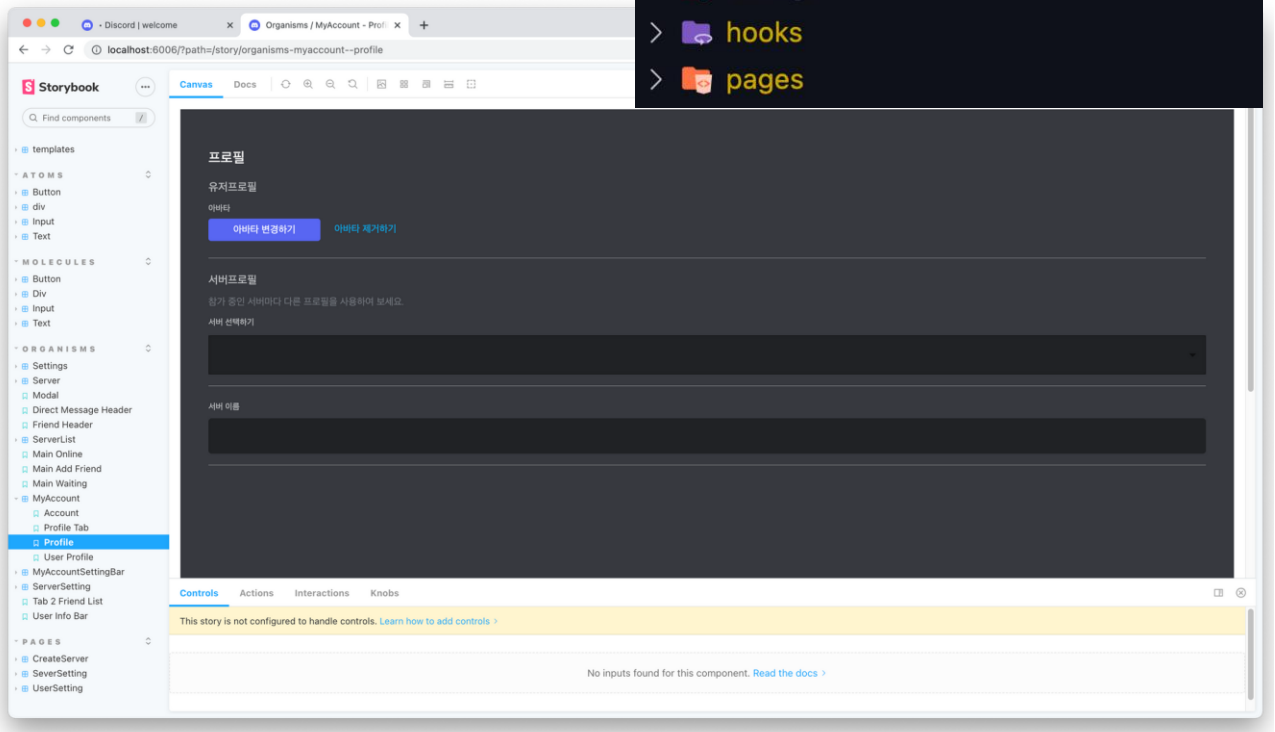
Design pattern + Storybook 적용

👇

Atomic Design 패턴 적용

Atomic Design 디자인 패턴에 따른 디렉토리 별 Storybook 적용 완료





개인목표 - 김현우

GOAL

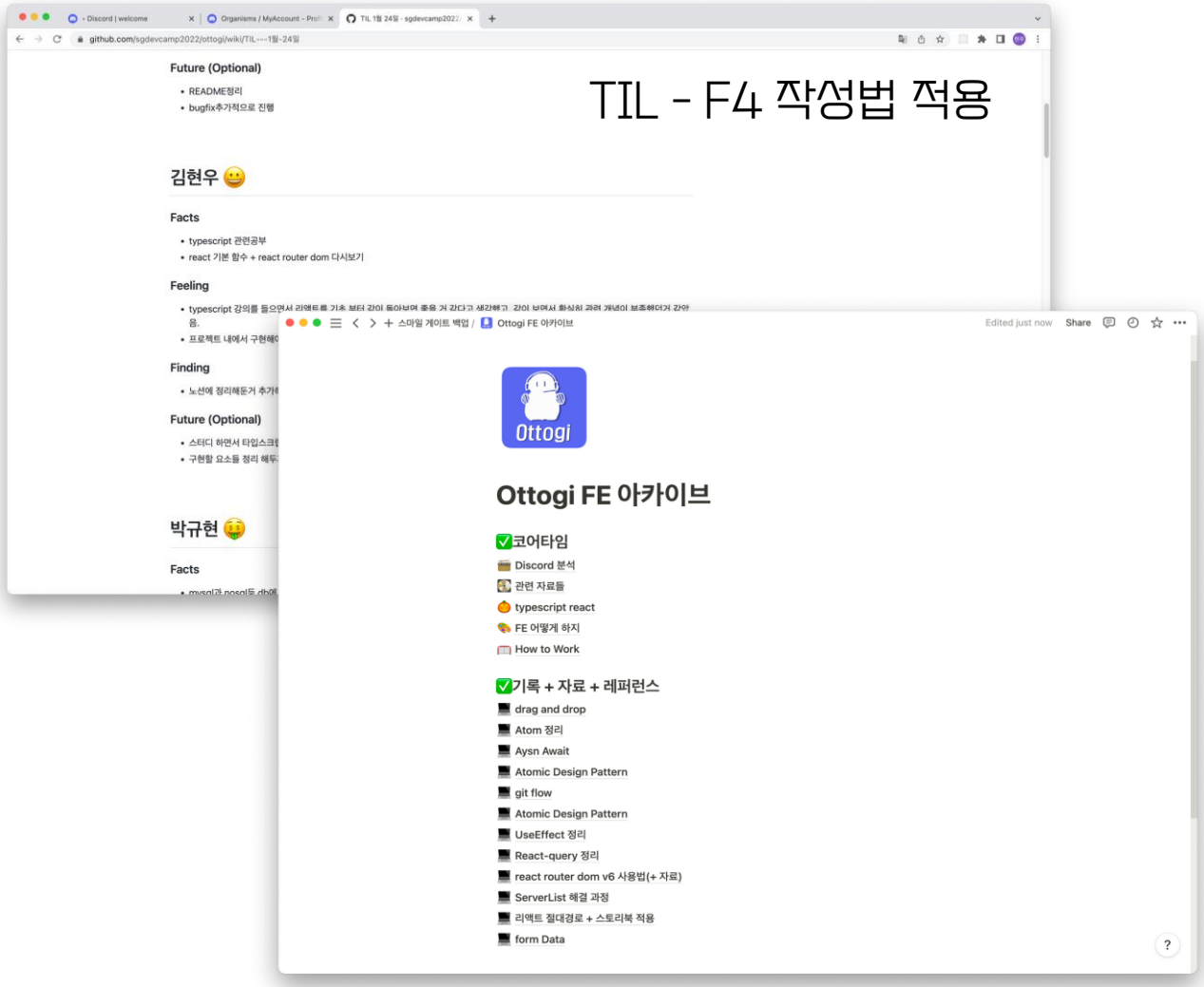
꾸준한 기록, 회고하는 습관 만들기

👇

팀 프로젝트 기간(약 6주간) TIL 작성 완료

관련 레퍼런스, 문제 해결 과정 기록

TIL - F4 작성법 적용



달성결과 - 김현우

이룬 점

- ✓ Typescript, Zustand, react-query, storybook 등 새로운 기술 스택에 도전
- ✓ TIL을 팀프로젝트 기간(약 6주간) 동안 꾸준히 작성
- ✓ Atomic Design Pattern을 적용해 기초부터 세세하게 프로젝트를 쌓아 올려볼 수 있었음.
- ✓ 프로젝트를 진행하면서 있었던 이슈, 문제에 대해서 그리고 해결에 필요한 레퍼런스를 정리

아쉬운 점

- ✓ Drag and Drop 기능을 미완성
- ✓ GitHub Issue 기능을 세세하게 사용하지 못했음
- ✓ 작성했던 글들에 대해서 블로그 업로드 까지 진행하지 못했음
- ✓ Webrtc 파트 미완성

FE/허다은

- 아토믹 디자인 패턴을 이용한 컴포넌트 구현
- 로그인/회원가입 페이지 구현
- 메인 페이지 구현
  - 친구 관리(요청, 추가, 삭제, 취소) 및 필터
  - 1대1 실시간 채팅 구현
- 커뮤니티 페이지 구현
  - 커뮤니티 실시간 채팅 구현
  - 커뮤니티 입장 및 퇴장 관리

절대 경로

문제점

상대 경로로 인해
import가 복잡하고 지저분해짐

절대 경로로 변경하려면 eject를 사용해서 따로 작성할 수 있으나, 많이 복잡함

→

해결책

craco !
(cra configuration operator)

절대 경로

```
yarn add @craco/craco
yarn add craco-alias -D
```

```
const CracoAlias = require("craco-alias");

module.exports = {
  plugins: [
    {
      plugin: CracoAlias,
      options: {
        source: "tsconfig",
        tsConfigPath: "tsconfig.paths.json",
      },
    },
  ],
};
```

```
{
  "compilerOptions": {
    "baseUrl": ".",
    "paths": {
      "@assets/*": ["src/assets/*"],
      "@api/*": ["src/api/*"],
      "@components/*": ["src/components/*"],
      "@configs/*": ["src/configs/*"],
      "@fonts/*": ["src/fonts/*"],
      "@hooks/*": ["src/hooks/*"],
      "@pages/*": ["src/pages/*"],
      "@store/*": ["src/store/*"],
      "@styles/*": ["src/styles/*"],
      "@utils/*": ["src/utils/*"]
    }
  }
}
```

타입스크립트도 고려하여 절대 경로를 세팅함

절대 경로

문제점

이후 스토리북에서 원인불명의
에러들이 다수 발생

→

해결책

스토리북도 따로 절대 경로 설정을
해야 하는거였군!

개인 결과물 - 허다은

절대 경로

```
{
  "compilerOptions": {
    "baseUrl": ".",
    "paths": {
      "@assets/*": ["src/assets/*"],
      "@api/*": ["src/api/*"],
      "@components/*": ["src/components/*"],
      "@configs/*": ["src/configs/*"],
      "@fonts/*": ["src/fonts/*"],
      "@hooks/*": ["src/hooks/*"],
      "@pages/*": ["src/pages/*"],
      "@store/*": ["src/store/*"],
      "@styles/*": ["src/styles/*"],
      "@utils/*": ["src/utils/*"]
    }
  }
}
```

```
const TsconfigPathsPlugin = require("tsconfig-paths-webpack-plugin");

module.exports = {
  ...
  webpackFinal: async (config) => ({
    ...config,
    resolve: {
      ...config.resolve,
      ...config.resolve.plugins.push(new TsconfigPathsPlugin({})),
    },
  }),
};
```

./storybook 폴더에서 절대 경로 세팅함

공통 headers

문제점

대다수의 api의 headers에
accessToken을 일일히 넣어줌

→

해결책

[멘토님 피드백 반영]

axios의 interceptors !

공통 headers

```
1  const accessToken = localStorage.getItem("accessToken");
2
3  clientApi.interceptors.request.use((config) => {
4    config.headers.Authorization = "Bearer " + accessToken;
5    return config;
6  });
```

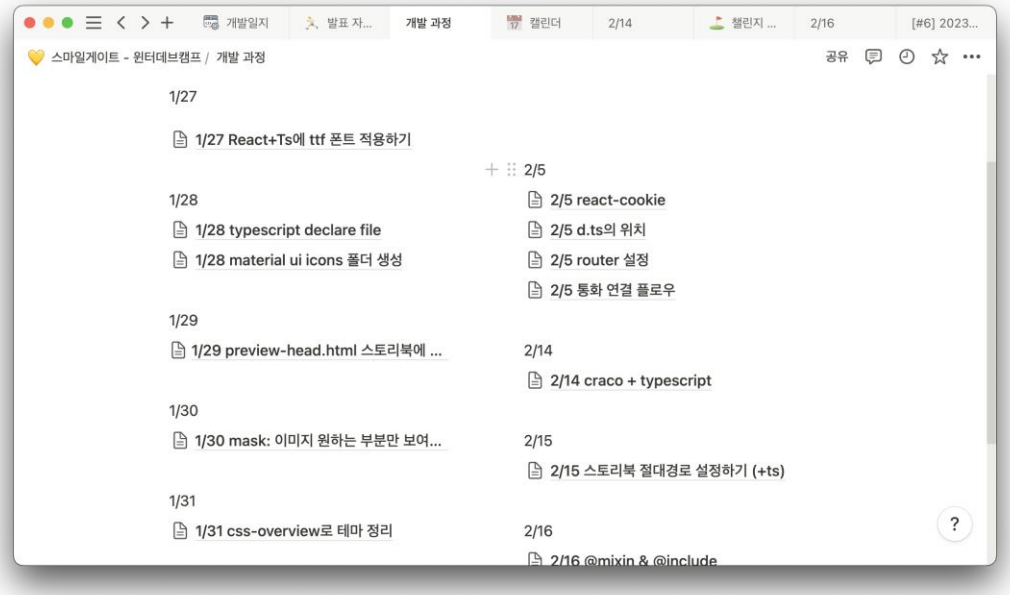
코드를 추가함으로써
각 api마다 추가해준 다량의
headers 코드를
제거할 수 있게 됨

개인목표 - 허다은

GOAL

이슈 해결과정의 문서화를 통한
이슈 관리법 정립

이슈 정리 및 해결방안 기록하기

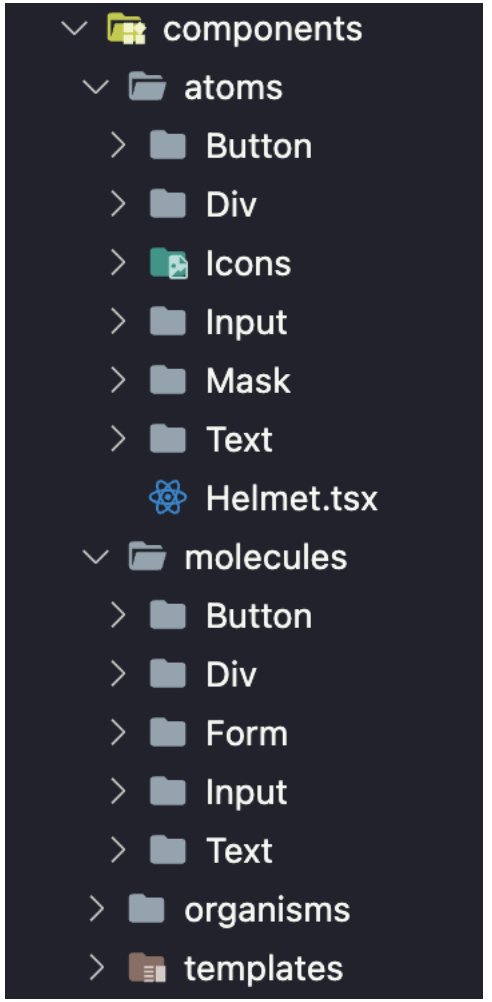


개인목표 - 허다은

GOAL

디자인 패턴 적용,
효율적 디렉토리 구조 설계

아토믹 디자인 패턴 적용하여
세세하게 프로젝트 구조 설계



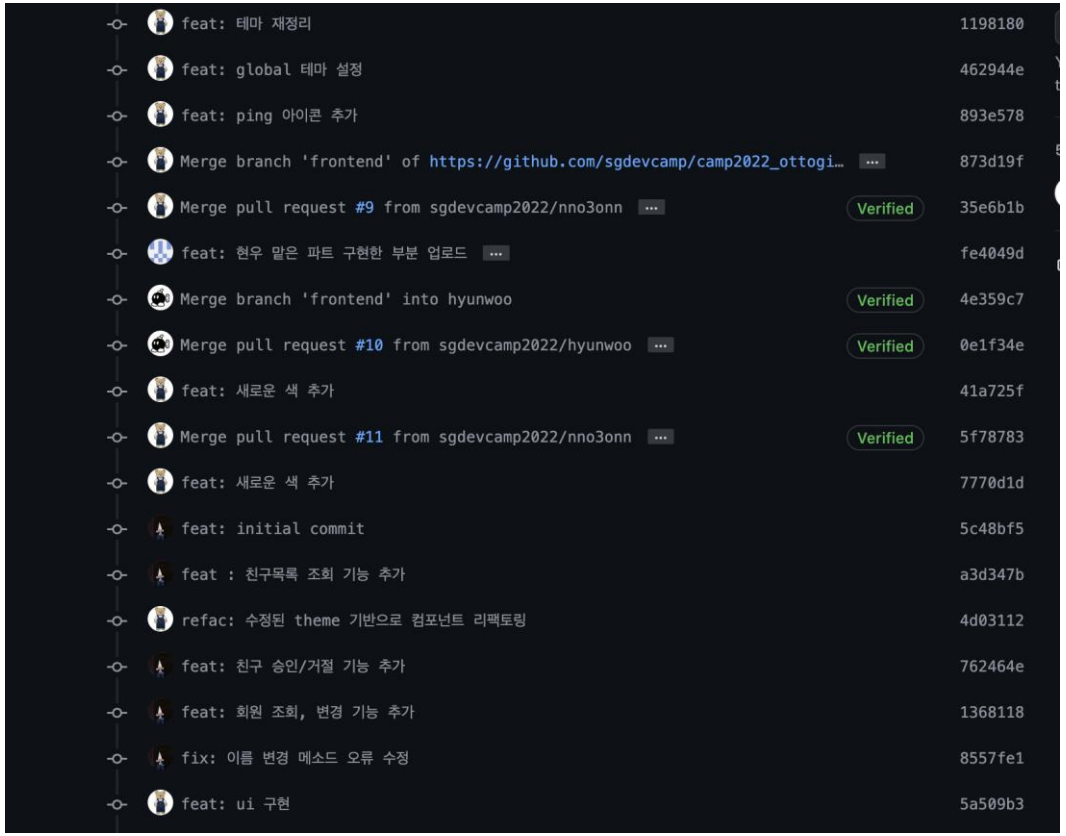
개인목표 - 허다은

GOAL

Git 커밋 컨벤션, git-flow 사용

👇

협업을 위한 깃허브 사용법 갖추기



달성결과 - 허다은

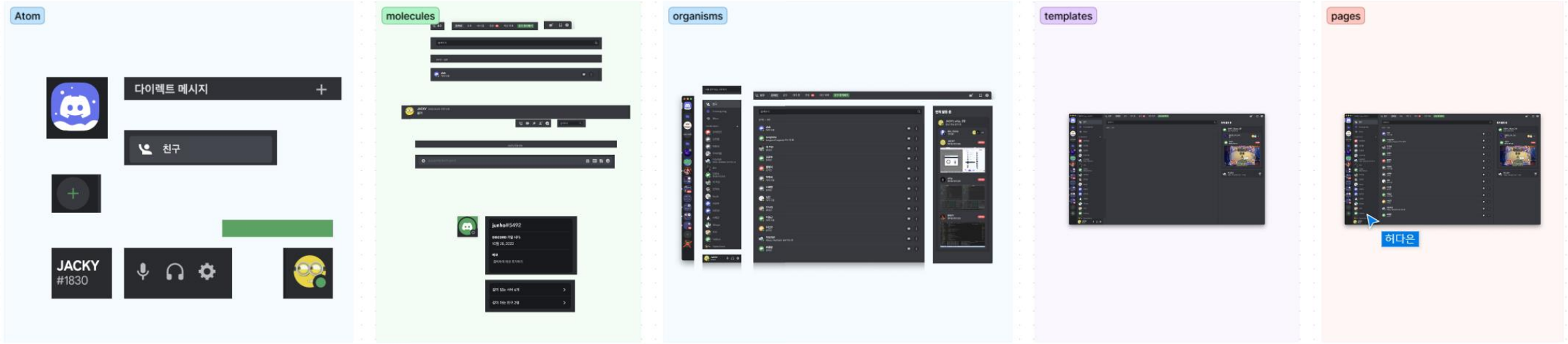
이룬 점

- ✓ 아토믹 디자인을 적용하여 처음으로 직접 탄탄하게 컴포넌트를 설계해보면서 설계의 중요성을 깨달음
- ✓ Zustand, react-query, storybook 등 새로운 기술 스택에 도전
- ✓ 나의 4F(Facts, Feeling, Finding, Future)을 담은 TIL을 매일 빠짐없이 기록함

아쉬운 점

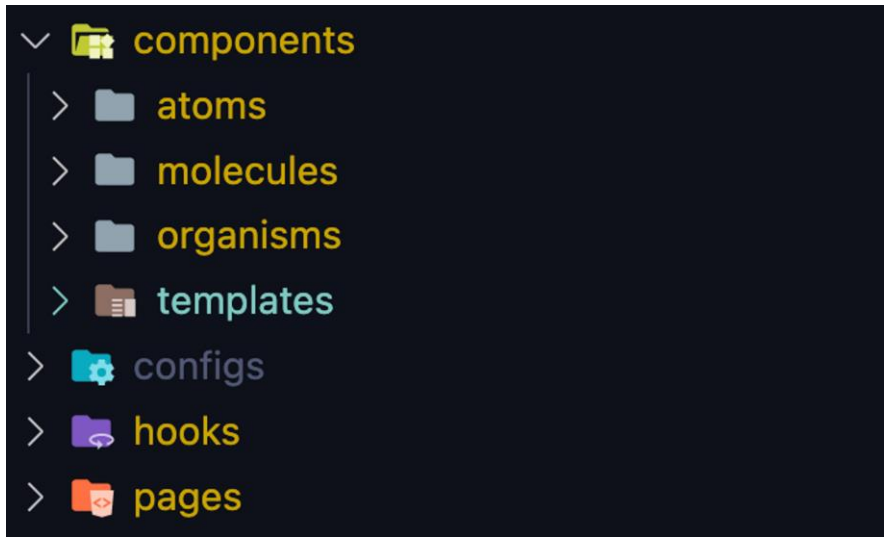
- ✓ User-flow를 좀 더 세세하게 분석하고 구현으로 넘어갔으면 flow 헤매지 않았을 것
- ✓ 초기에 계획했던 모든 기능을 구현하지 못하였음
- ✓ Storybook의 action 기능 활용 미흡
- ✓ 팀원들과 GitHub의 issue, PR 기능을 활용하지 않아 코드 리뷰를 하지 못해 아쉬웠음
- ✓ 액세스 토큰 만료 시 리프레시 토큰과 액세스 토큰을 재발급하는 과정을 끝까지 구현하지 못하였음

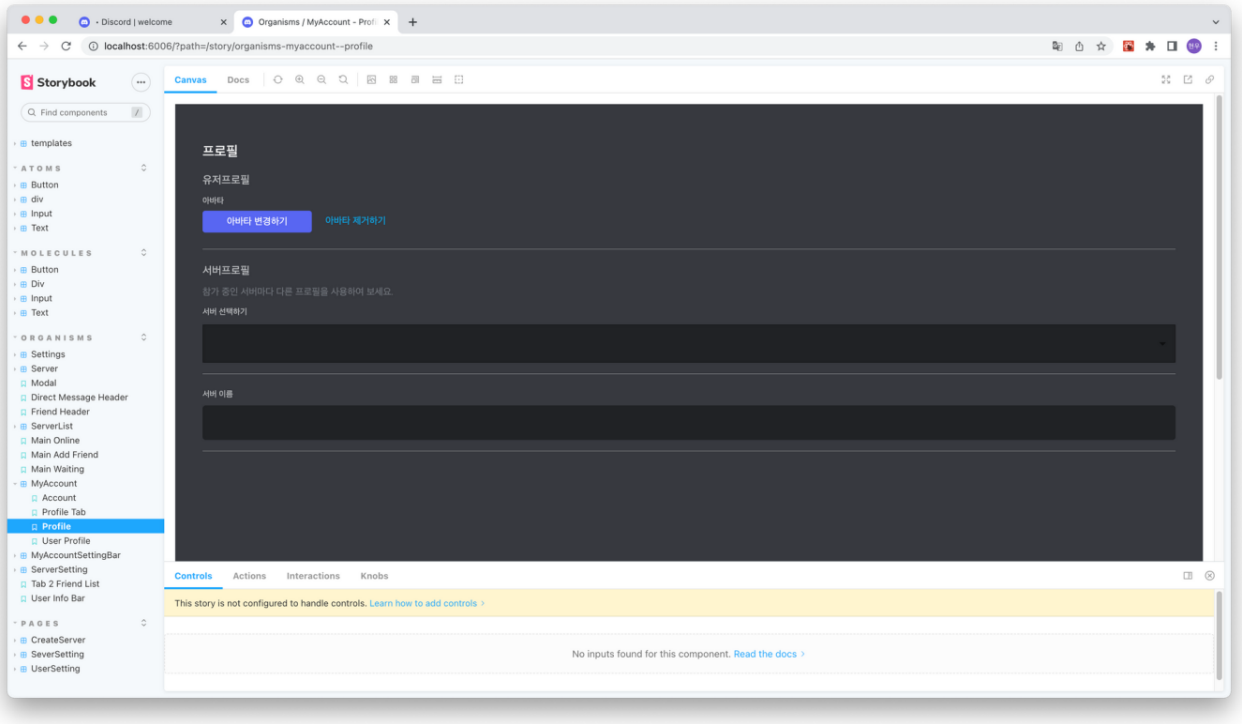
FE : Atomic Design Pattern & Storybook



Atom 👉 Molecule 👉 Organism 👉 Template 👉 Pages 순으로
Atomic Design Pattern을 적용하여 프로젝트 진행

Atomic Design Pattern & Storybook





React Query

```tsx
const { data, isSuccess } = useGetFriendList(email);
  if (!isSuccess) return <></>;

  const friendList: FriendType[] = data.filter(
    (friend: FriendType) => friend.friendState === "ACCEPTED"
  );
```

→

```tsx
import { useQuery } from "@tanstack/react-query";
import friendApi from "@api/friend";

const useGetFriendList = (email: string) => {
  const { data, isSuccess } = useQuery(
    ["friendList", { email }],
    friendApi.getAll
  );

  return { data: data?.data.data, isSuccess };
};

export default useGetFriendList;
```

→

```tsx
const friendApi = {
  getAll: async ({ queryKey }: any) => {
    const { email } = queryKey[1];
    return await clientApi.get(`/user/member/showfriend`, {
      params: { email },
    });
  },
```

useGetFriendList - useQuery - friendApigetAll과 같이

호출, react-query, api 순으로 3단계를 거쳐 서버 요청 진행

Zustand

- ✓ 비동기 사용의 편이
- ✓ Persist와 같은 툴 설치 필요 X
- ✓ Redux에 비해 적은 보일러 플레이트

```tsx
interface TabState {
  mainStatus: MainStatusType;
  userId: number;
  userName: string;
  deleteFriendEmail: string;
}
interface TabAction {
  setMainStatus: (mainStatus: MainStatusType) => void;
  setUserId: (userId: number) => void;
  setUserName: (userName: string) => void;
  setDeleteFriendEmail: (deleteFriendEmail: string) => void;
}
const useMainStore = create<TabState & TabAction>()(
  devtools(
    persist(
      (set) => ({
        mainStatus: "온라인",
        userId: -1,
        userName: "",
        deleteFriendEmail: "",
        setMainStatus: (mainStatus: MainStatusType) => set({ mainStatus }),
        setUserId: (userId: number) => set({ userId }),
        setUserName: (userName: string) => set({ userName }),
        setDeleteFriendEmail: (deleteFriendEmail: string) =>
          set({ deleteFriendEmail }),
      }),
      { name: "main" }
    )
  )
);
```

Typescript

- ✓ Typescript를 사용함으로써 에러를 실시간으로 확인해 사전에 방지하였음
- ✓ Typescript Declare 파일에 서버에서 받아온 데이터 타입을 선언해 둠으로써 개발 생산성을 향상 할 수 있었음

```typescript
interface UserInfoType {
  id: number;
  email: string;
  name: string;
  introduction: string;
  profileImagePath: string;
  createdAt: string;
}

type FriendStateType = "REQUEST" | "WAIT" | "ACCEPTED";

interface FriendType {
  userId: number;
  name: string;
  email: string;
  friendState: FriendStateType;
  channelId: string;
  createdAt: Date;
  profileImagePath: string;
}
```

회고

- ✓ Zustand, React-query와 같은 상태관리 라이브러리를 통해 데이터를 효과적으로 관리 해볼 수 있었고, 이전에 사용했던 Redux와 비교해볼 수 있었고, 상황에 맞는 라이브러리의 선택할 수 있는 능력을 키울 수 있었다.
- ✓ 프론트를 맡은 팀원 모두 Typescript를 처음 사용해서 익숙하지 않아서 오류가 발생할 때 마다 많이 헤맸지만 Typescript의 적용을 통해서 에러와 버그를 사전에 예방하면서 코드를 작성할 수 있어서 좋았다.
- ✓ 두 명이서 프론트를 함께 진행하면서 혼자서 해결하기 힘들었던 문제들을 함께 고민하며 빠르게 해결할 수 있었고, 팀원 간의 다른 관점을 공유함으로써 코드를 바라보는 시야를 넓힐 수 있었다.
- ✓ 디스코드에 Atomic Design Pattern을 적용해보면서 프로젝트 설계의 중요성을 알 수 있었고, 기초 부터 쌓아 올라가는 것이 얼마나 중요한지 깨닫게 되었다.

개인 결과물 - 김수찬

BE / 김수찬

- WebRTC
  - 영상 송신
    - Publish & Subscribe
  - 영상 수신
  - Signaling Server

✓영상을 전송 하는 방식

✓ Publish 진행

Publish 버튼 클릭

Connect →

Media 정보 전송


```
{
  "id": "00ee8229-2a68-41ff-b311-f88a6135636c",
  "iceParameters": {
      "iceLite": true,
      "password": "0x48gj5s3pe4qsjwpext5oadceb1r5kl",
      "usernameFragment":
"qjmmatretvsycqhufcjzjycrq8tb1be7"
  },
  "iceCandidates": [ ... ],
  "dtlsParameters": { ... }
}
```


→ Signaling Server로 유저 정보 저장

✓ Subscribe 진행

저장 받은 정보 상대방에게 에게 전송 → Media 정보 바탕으로 Media Track 형성 → 영상 연결

✓영상 송신

Server와 연결을 진행
connectSocket()

비동기 방식을 이용하여

서버에 영상 및 음성 정보 제공할
Transport 생성

이후, 이 영상정보를 Publish 하여,
이후 접속하는 consumer 에게
전달할 준비를 진행

```
async function handleConnect() {

    // --- connect socket.io ---
    await connectSocket().catch((err: any) => { });

    // --- get capabilities --
    const data = await sendRequest('getRouterRtpCapabilities', {
        roomName : roomName,
    });
    await loadDevice(data);
    ...
    // --- get transport info ---
    const params = await sendRequest('createProducerTransport', {
        mode: MODE_STREAM,
    });
    ....
    // --- join & start publish --
    producerTransport.current.on('connect');
    producerTransport.current.on('produce');
   ....

    if (useVideo) {
        const videoTrack = localStream.current.getVideoTracks()[0];
        ...
    }
    if (useAudio) {
        const audioTrack = localStream.current.getAudioTracks()[0];
        ...
    }
}
```

개인 결과물 - 김수찬

✓영상 수신

Server가 제공받은 음성정보를
request
consumeAdd()

addConsumer에서
상대방이 Server에 제공한
영상 및 음성정보를 Track에 올림

이 과정을 비동기 방식으로 진행하여,
Delay 최소화

```
async function consumeAdd(
    transport: any,
    remoteSocketId: any,
    prdId: any,
    trackKind: any,
    mode: any = MODE_STREAM
) {
    const { rtpCapabilities } = device.current;
    const data = await sendRequest('consumeAdd', { })
    const consumer = await transport.consume({   });

    addConsumer(remoteSocketId, consumer, kind, mode);
    consumer.remoteId = remoteSocketId;
    consumer.on('transportclose', () => {   });
    consumer.on('producerclose', () => {
        consumer.close();
        removeConsumer(consumer.remoteId, kind, mode);
        removeRemoteVideo(consumer.remoteId, mode);
    });

    if (kind === 'video') {
        console.log('--try resumeAdd --');
        sendRequest('resumeAdd', {
            remoteId: remoteSocketId,
            kind: kind,
            mode,
        })}
    return new Promise((resolve: any, reject: any) => {
        addRemoteTrack(remoteSocketId, consumer.track, mode);
        resolve();
    });
}
```

개인목표 - 김수찬

최대한 구체적이지만, 간단하게 작성

GOAL 80%

컨벤션 및 합의된 변수명을 활용,
협업에 용이하게 프로그래밍 방식 변환
git commit을 직관적으로 작업

개인목표 - 김수찬

적는다고 적긴 했는데 많이 아쉽다.

GOAL 65%

Markdown을 활용한 스펙 설명
(기능 설명 및 버전 관리)
필수적인 요소
(필요한 input 및 output 제시)

1. 역할

서비스	역할
`미디어 서버`	1. SFU 방식의 서버를 사용하여 **클라이언트의 부하 감소** 2. WebRTC를 이용한 화상채팅 및 음성채팅 제공
`시그널링 서버`	1. 사용자간의 연결성 파악 2. 사용자의 현재 위치정보를 제공 및 저장

2. 기술스택

- Node.js 18.12
- socket.io
- socket.io-client
- express
- WebRTC
- Mediasoup v3
- Mediasoup-client
- bundle.js
- react
- browserify
- `DB는 아직 정하지 못했음`

  실시간 테스팅을 위해 사용
  - watchify

Server 쪽

현재 서버 쪽에서 NoSQL 방식으로 저장하는 data는 총 5가지

데이터	설명
Rooms	존재하는 룸의 {Router, 및 peers의 ip}
Peers	Peer의 IP 정보에 따른, Room의 위치, socket 그리고 peer들의 정보 (`Workers의 id인지 제공자의 id인지는 확인해봐야함`)
Transports	:-------
Producers	Client가 접속했을 때, producers의 정보
Consumers	Client가 meida를 제공해줘야할 Consumer의 정보

1. 방 입장

client

```
const joinRoom = () =>{ // to make router or go to router
  socket.emit('joinRoom', {roomName}, (data) => {
    ...
    createDevice()
  })
}
```

- joinRoom을 emit 이후 CreateDevice() 진행

server

```
socket.on('joinRoom', async({roomName}, callback) =>
{
  // create router if room is note exist
  ... 방 생성

  }
  // get Router RTP capabilities
```

개인목표 - 김수찬

가장 많이 득을 봄.. 갈아 엎을 때 도움이 많이 되었다.

GOAL 90%

Function들의 기능 세분화
-> 유지보수가 용이

정말로 유지보수가 용이했다! 리펙토링까지..

```
// Server 부분
run();
async function run() {
    try {
        await createMediasoupWorkers();

        await runExpressApp();

        await runWebServer();

        await runSocketServer();

    } catch (err) {
        console.error(err);
    }
}
```

```
//Client 부분
UI 부분 ========
const handleStartScreenShare = () => { // 화면공유
...
  const mediaDevices: any = navigator.mediaDevices;
...
}

const handleStartMedia = async() => { // 화상 대화
  ...
    navigator.mediaDevices
        .getUserMedia({ audio: useAudio, video: useVideo })
  ...
}

function stopLocalStream(stream: any) { // 영상 중지
    let tracks = stream.getTracks();
    if (!tracks) {
        console.warn('NO tracks');
        return;
    }
    tracks.forEach((track: any) => track.stop());
}

function removeRemoteVideo(id: any, mode: string) { // Peer 영상 제거
  ...
    delete consumersStream.current[id];
  ...
}

Socket 통신 부분 ==========
```

달성결과 - 김수찬

이룬 점

- ✓ 웹에 대한 다양한 경험을 짧은 시간내에 할 수 있었음
- ✓ 프로젝트 중간에 아키텍처 변경으로 인하여 오히려 FE와 BE에 구분에 대하여 구체화됨
- ✓ PMP를 통하여 기획에 대한 구체성 있는 경험

아쉬운 점

- ✓ PMP 단계에 비하여 개발 중에는 교류가 부족했기에 팀원간 피드백에 대한 부분이 아쉬웠음
- ✓ JavaScript를 처음 사용해보며 https와 socket.io에 대한 지식 부족
- ✓ 아키텍처 설계에 대한 구체화가 부족 했었음

느낀점 및 향후 목표

1. 캠프를 갈무리 할 때쯤, 아키텍쳐 설계를 다시 한 번 확인 했는데, 설계 및 방법론적인 부분에서 내용이 많이 약했던 것 같다.

2. 무엇을, 왜, 어떻게 라는 부분을 더욱 구체적으로 작업을 하고 이야기를 나누어야 함을 배움
   정보 전달 및, 업무의 구체성을 더욱 잡아낼 수 있을 것이라 보이며, 이것을 지금에서야 깨달은게 많이 아쉽다.

3. 부족한 시간안에
   a. 기술을 선정하는
   b. 기술을 이용하는
   c. 작업을 엎어야 하는, 많은 일들이 있었다.

2개월이라는 개발 기간 동안,
처음 웹을 다뤄보면서 인상적인 일,
특히 선택과 관련된 일들을 직접 겪고 다뤄보며
팀원과 나아갔던 것이 이번 캠프에서
가장 크게 얻어간 것 같다.

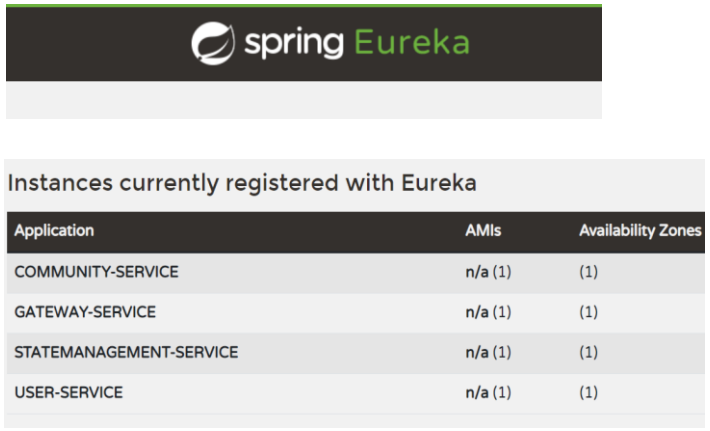
BE / 백종인

- MSA 구조 적용
  - Spring Cloud 사용
- JWT 활용 인증 서버 구현
- Web Socket 기반 채팅 서비스 구현
  - Stomp, RabbitMQ 사용
- AWS S3를 통한 파일관리

개인 결과물 - 백종인

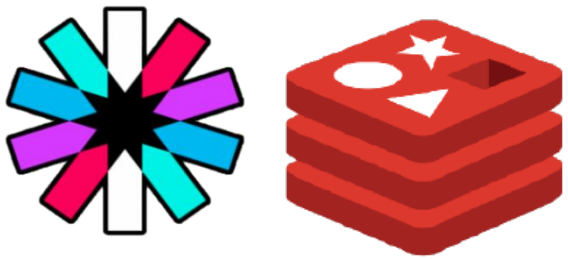
1. MSA 구조를 처음 사용하여 프로젝트 진행

- 다양한 언어로 작성된 서비스를 등록해서 관리



2. JWT 를 이용한 인증 구현

- 기존에는 세션만 이용하다가 JWT 인증을 구현
- 인증서버에서 토큰 발급 및 Gateway에서 JWT 인증 절차 거침



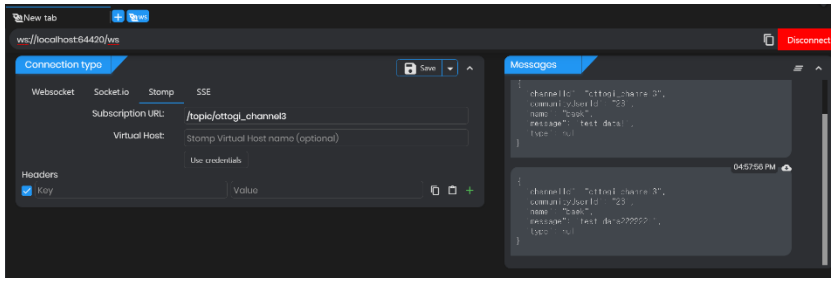
개인 결과물 - 백종인

3. Web Socket 기반 서비스 구현

- 실시간 채팅 서비스를 위하여 STOMP 기반 pub/sub 채팅서버 구현
- Rabbit MQ 외부 메시지 브로커를 적용하여 다중 서버에도 대응
- 비정형 데이터를 Redis에 저장하여 관리 용이

4. AWS S3 사용

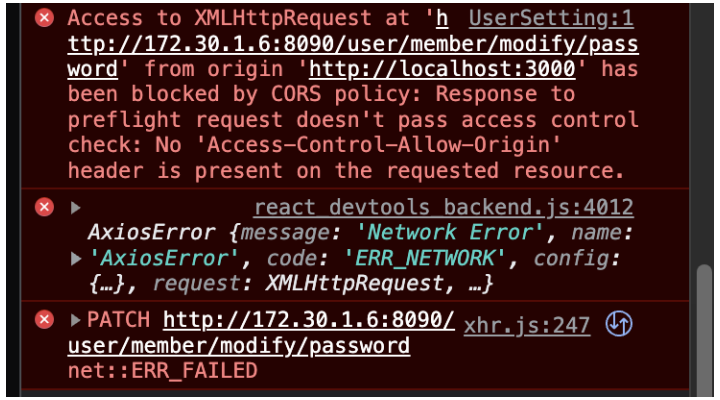
- 유저서버에서 유연한 파일 관리를 위한 S3 사용



고민 및 어려웠던 부분 - 백종인

Cors 및 설정 문제

- 프론트와 처음 합쳐보면서 만났던 문제
- 개념이 부족했고, API Gateway구조에서 발생했기에 조금 더 까다로 웠음
- 특정 HTTP 메소드가 허용되지 않는 문제



해결

- Gateway와 Security 부분에서 특정 IP를 허용해줌
- HTTP Method 또한 따로 허용해주는 세팅을 함으로 해결

```
Bean
orsConfigurationSource corsConfigurationSource() {
    CorsConfiguration configuration = new CorsConfiguration();
    configuration.setAllowedOrigins(Arrays.asList("https://ottogi-ottogi.vercel.app", "http://localhost:30
    configuration.setAllowedMethods(Arrays.asList("HEAD", "GET", "POST", "PUT","OPTIONS"));
    configuration.setAllowedHeaders(Arrays.asList("Authorization", "Cache-Control", "Content-Type"));
    configuration.setAllowCredentials(true);
    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration( pattern: "/**", configuration);
    return source;
```

고민 및 어려웠던 부분 - 백종인

Node JS Service 등록

- Spring Cloud를 사용하면서 Node Service 등록에 이슈
- 처음에 Spring Cloud Sidecar 를 사용하려 했으나,
  원하는 대로 설정하지 못함

해결

- Eureka js client를 사용하여 Node 쪽에서 서비스를
  등록하는 방식으로 해결

3.2. Sidecar Application

First, we need to have a Eureka Server up. After Eureka Server is started, we can ac
http://127.0.0.1:8761

Let's add *spring-cloud-netflix-sidecar* as a dependency:

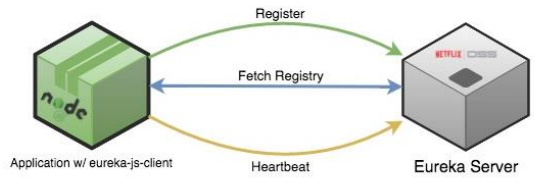
```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-netflix-sidecar</artifactId>
    <version>2.2.10.RELEASE</version>
</dependency>
```

eureka-js-client

npm package 4.5.0 | build passing | coverage 97% | Dependency Status

bitHound Overall Score

A JavaScript implementation of a client for Eureka
(https://github.com/Netflix/eureka), the Netflix OSS service registry.



고민 및 어려웠던 부분 - 백종인

실시간 채팅 위치 정보 받아오기

- 채팅방 입장 시, 어느방에 들어오는지 체킹하기가 어려웠음
- 소켓 연결 및 해제시 기본적인 이벤트는 체킹할 수 있었으나 클라이언트 정보 받아오는데에 문제

```java
@Slf4j
@Component
public class WebSocketChatEventListener {

    @EventListener
    public void handleWebSocketConnectListener(SessionConnectedEvent event){
        log.info("새로운 소켓 연결");

    }

    @EventListener
    public void handleWebSocketDisconnectListener(SessionDisconnectEvent event) {
        log.info("소켓 연결 해제");
    }
```

해결

- 프론트에서 소켓 연결을 요청할 때 헤더 부분에 데이터를 넣어주고 연결하고 서버쪽에서 인터셉터를 만들어서 파싱
- 이후 상태관리 서버로 데이터를 보내주는 방식으로 채팅 위치 관리

```java
@Override
public Message<?> preSend(Message<?> message, MessageChannel channel) {

    StompHeaderAccessor stompHeaderAccessor = StompHeaderAccessor.wrap(message);

    if(stompHeaderAccessor.getCommand() == StompCommand.CONNECT){
        String userId = stompHeaderAccessor.getFirstNativeHeader( headerName: "userId");
        String channelId = stompHeaderAccessor.getFirstNativeHeader( headerName: "channelId");
        String sessionId = (String) message.getHeaders().get("simpSessionId");
```

달성결과 - 백종인

이룬 점

- ✓ 여태 해보지 못했던 MSA 구조의 아키텍처로 개발
- ✓ 팀원들과 깊은 협업 및 프론트와 합쳐보는 작업
- ✓ 다양한 기술 스택을 처음 공부 및 구현하면서 공부한 내용을 문서화
- ✓ 공부 -> 구현 과정을 반복적으로 거치며 구현에 대한 자신감 상승

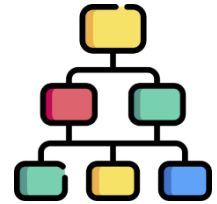
아쉬운 점

- ✓ 목표한 테스트 코드를 작성하지 못함
- ✓ 본격적인 협업 프로젝트를 처음 해보아서 스케쥴 관리나 소통에 있어서 부족한 점이 많았음

개인 결과물 - 박규현

BE / 박규현

1. 계층형 커뮤니티 서버 구현

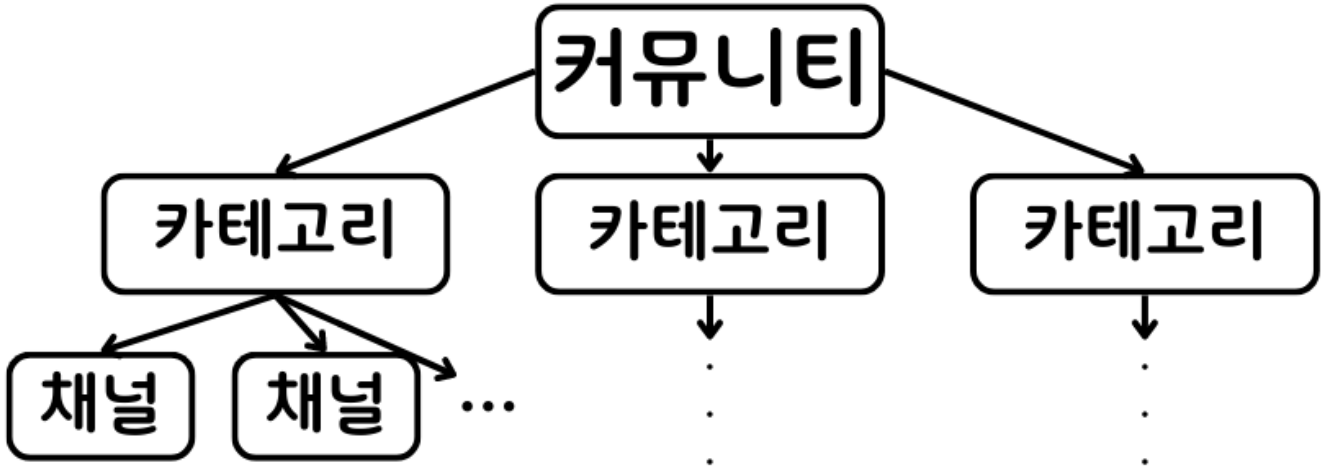


- 커뮤니티 > 카테고리 > 채널로 이어지는 계층형 커뮤니티 구성
- 커뮤니티가 삭제되면 카테고리, 채널이 연쇄적으로 삭제되는 시스템
- shortURL 을 활용하여 초대장 로직 생성

2. 상태관리 서버

- 다양한 서버로부터 받아오는 정보들을 관리하는 서버 구현

✓상위 계층과 하위 계층의 연계성



상위 커뮤니티에 카테고리가 묶여 있고, 카테고리에 다시 채널이 묶여 있는 형태

✓상위 계층과 하위 계층의 연계성

기존 해결 방안 ⟶ 코드 리뷰 이후 해결 방안

1. 상위 계층이 삭제되면 하위 계층도 삭제됨
2. 상위 계층의 정보를 통해 하위 계층을 조회
3. 조회된 하위 계층의 정보로 다시 하위 계층 조회

=> 상위 계층과 하위 계층이 단단한 결합이 되어있고, 여러 번의 조회 쿼리문을 사용하여 비효율적임

1. 상위 계층이과 하위 계층을 외래키로 묶는다
2. 묶인 것을 기준으로 상위 계층이 삭제되면 하위 계층도 같이 삭제됨

=> 상위 계층과 하위 계층의 결합이 비교적 가벼워지고, 비효율적인 쿼리문 사용 빈도를 줄일 수 있었음

✓초대장을 처리 하는 방식

✓ 초대권 부여

초대 버튼 클릭

START

→

정보 담아서 요청

```
{
    "communityId":"1",
    "userId":"1",
    "shortUrl":"dP6G4KPZ"
}
```

→

커뮤니티 서버에 해당 정보와 위치(channel_id) 저장

✓ 초대 링크 접속

생성된 링크 접속

`'http://localhost:3000/invite/dP6G4KPZ/1'`

→ 커뮤니티 서버 정보 조회 → 커뮤니티 참가

✓이미지 업로드 방식

이미지를 등록하면 multer를 이용하여 s3에 업로드 -> s3에 업로드 된 이미지 url을 받아와서 DB에 저장 하는 방식

multer

Amazon S3

✓상태관리 서버 - 실시간 처리가 필요한가?

A : 채팅에서 실시간 정보를 받아서 요청하는 것이기 때문에 굳이
실시간으로 할 필요가 없을 것이라고 판단

로그인 / 로그아웃
채팅방 접속 / 해제 ⟶ 상태 관리 서버에 업로드

```
{
    "channelId":null,
    "userId":"1",
    "status":"0",
    "sessionId":"~~~"
}
```

⟶ 상태 정보와 위치 (channel) 정보 업데이트

달성결과 - 박규현

이룬 점

- ✓ MSAV 패턴을 참고하여 도메인 로직과 UI를 분리
- ✓ 변수명과 함수명을 직관적으로 사용하려고 노력
- ✓ 아키텍처를 먼저 생각해보고 코드를 작성
- ✓ Node js를 포함한 여러 백엔드 스택에 이해도가 늘음
- ✓ PMP를 작성하면 설계에 대한 심도 깊은 경험
- ✓ 코드리뷰, PMP리뷰로 부족한 점을 알 수 있었음

아쉬운 점

- ✓ 회고록, TIL을 작성하는 것이 많이 부족함
- ✓ 테스트 코드를 도전해볼 시간이 부족했음
- ✓ 성능 최적화 단계를 경험해보지 못하여 아쉬움

팀 결과물

팀 목표

“TEAM GOAL 🎯”

1. 중요한 것은 꺾이지 않는 마음!

- 캠프 기간 동안 중도탈락 없이 프로젝트 끝까지 마무리하기

2. 기억은 희미해지지만 기록은 희미해지지 않는다.

- 이슈 상황이 생겨서 문제 해결을 하거나 구현을 위해 필요한 지식을 학습한 일련의 과정들에 대해서 문서화하여 회고 할 수 있는 자료 만들기

3. 도전은 경험을, 경험은 기회를

- 경험이 없는 아키텍처, 디자인 패턴에 대해서 이해하고 적용하기

팀 목표

“TEAM GOAL 🎯”

1. 중요한 것은 꺾이지 않는 마음! ✅

-> 팀원 중 중도하차 한 명 없이 프로젝트 완료!

2. 기억은 희미해지지만 기록은 희미해지지 않는다. ✅

-> 매일 TIL 작성으로 학습한 내용을 꾸준히 기록

3. 도전은 경험을, 경험은 기회를 ✅

-> 기존의 BE는 FE, FE는 BE 도전 했고 ,
각자 새로운 기술 스택을 사용하여 프로젝트 수행!

END

감사합니다.