Zhzzk - 스트리머 플랫폼 토이 프로젝트

프로젝트 개요

Zhzzk는 치지직, 트위치, 유튜브와 같은 다양한 플랫폼 기반의 **스트리밍 방송 서비스 플랫폼**입니다.
이 프로젝트는 사용자가 플렛폼 별로 스트리머 정보를 검색하고, 방송을 시청하며, 실시간 채팅을 할 수 있도록 설계되었습니다. 또한, WebRTC를 활용한 화면 공유 기능과 Firebase를 사용한 데이터 관리, CI/CD 자동화 배포를 통해 효율적인 개발 및 배포 환경을 구축하였습니다.

주요 기능

1. **다양한 스트리머 플랫폼 지원:** 치지직, 트위치, 유튜브 기반 스트리머 관리
2. **실시간 채팅:** WebSocket을 사용하여 방송 중 채팅 가능
3. **화면 공유:** WebRTC를 활용한 스트리머 화면 공유
4. **스트리머 데이터 관리:** Firebase Realtime Database 연동
5. **CI/CD 자동화:** Azure DevOps를 사용한 자동 빌드 및 배포

기술 스택

Frontend

- **Framework:** React 18.3.1, Next.js 15.0.2, typescript 5
- **CSS:** Tailwind CSS
- **Streaming:** Twitch Player SDK, WebRTC
- **State Management:** Zustand

Backend

- **Framework:** NestJS
- **Database:** Firebase Realtime Database
- **API:** REST API, WebSocket
- **Authentication:** Twitch OAuth

DevOps

- **CI/CD:** Azure DevOps Pipelines
- **Deployment:** Azure App Service

주요 코드 설명

Frontend

1 스트리머 목록 관리

파일: `streamerService.ts`
Firebase Realtime Database를 사용하여 스트리머 데이터를 관리합니다.

```
import { getDatabase, ref, get, set } from "firebase/database";

// 스트리머 추가 함수
export const addStreamer = async (streamers: Streamer[]) => {
  const db = getDatabase();
  const streamersRef = ref(db, "streamers");
  await set(streamersRef, streamers);
};

// 스트리머 조회 함수
export const getStreamers = async (): Promise<Streamer[]> => {
  const db = getDatabase();
  const snapshot = await get(ref(db, "streamers"));
  return snapshot.exists() ? Object.values(snapshot.val()) : [];
};
```

설명:

- addStreamer: Firebase에 스트리머 데이터를 저장합니다.
- getStreamers: Firebase에서 스트리머 목록을 가져옵니다.

2 WebRTC 기반 화면 공유

파일: `StreamPlayer.tsx` WebRTC를 활용하여 화면 공유 기능을 구현합니다.

```
const startScreenStreaming = async () => {
  const screenStream = await navigator.mediaDevices.getDisplayMedia({ video: true
});
  peerConnection.current = new RTCPeerConnection();
  screenStream.getTracks().forEach((track) => {
    peerConnection.current?.addTrack(track, screenStream);
  });
};
```

설명:

- getDisplayMedia: 화면 공유 스트림을 가져옵니다.
- RTCPeerConnection: WebRTC를 사용하여 P2P 연결을 설정합니다.

3 실시간 채팅

파일: `chat.gateway.ts` WebSocket을 사용하여 실시간 채팅을 구현합니다.

```
@WebSocketGateway()
export class ChatGateway {
  @WebSocketServer() server: Server;
```

```
  @SubscribeMessage('message')
  handleMessage(@MessageBody() data: { nickname: string; message: string }): void
{
    this.server.emit('message', { nickname: data.nickname, message: data.message
});
  }
}
```

설명:

- handleMessage: 클라이언트에서 전송된 메시지를 모든 클라이언트에게 브로드캐스트합니다.

4 Twitch 스트리머 연동

파일: `twitchService.ts` Twitch API를 통해 스트리머 정보를 가져옵니다.

```
export const getStreamerInfo = async (streamerName: string):
Promise<TwitchStreamerInfo> => {
  const response = await axios.get(
    `https://api.twitch.tv/helix/streams?user_login=${streamerName}`,
    { headers: { Authorization: `Bearer ${ACCESS_TOKEN}`, "Client-ID": CLIENT_ID }
}
  );
  return response.data.data[0];
};
```

설명:

- Twitch API를 통해 스트리머의 실시간 방송 정보를 가져옵니다.

5 CI/CD 파이프라인 설정

파일: `azure-pipelines.yml` Azure DevOps를 사용해 CI/CD 파이프라인을 설정합니다.

⚙ Backend (NestJS)

Zhzzk 프로젝트의 백엔드는 NestJS를 기반으로 구축되었으며, 다음과 같은 주요 기능을 제공합니다:

1. **실시간 채팅 (Socket.IO)**
2. **Twitch API 연동**
3. **치지직 API 연동**
4. **WebRTC 시그널링 서버**

---

1 실시간 채팅 (Socket.IO)

🗁 **파일:** `chat.gateway.ts` WebSocket을 사용해 실시간 채팅을 구현했습니다.
Socket.IO를 활용하여 클라이언트 간 메시지를 주고받습니다.

```
import { WebSocketGateway, SubscribeMessage, MessageBody, WebSocketServer,
OnGatewayInit, OnGatewayConnection, OnGatewayDisconnect } from
'@nestjs/websockets';
import { Server, Socket } from 'socket.io';
import { Logger } from '@nestjs/common';
import { ChzzkChatService } from './chzzkAPI';

@WebSocketGateway({
  cors: {
    origin: '*', // CORS 허용 설정
    credentials: true,
  },
})
export class ChatGateway implements OnGatewayInit, OnGatewayConnection,
OnGatewayDisconnect {
  @WebSocketServer() server: Server;

  private readonly logger = new Logger('ChatGateway');
  private chatServices: Map<string, ChzzkChatService> = new Map();
  private clientToStreamerMap: Map<string, string> = new Map(); // 클라이언트와 스
트리머의 매핑을 관리.

  afterInit(server: Server) {
    console.log('Socket server initialized');
  }

  handleConnection(client: Socket) {
    console.log(`Client connected: ${client.id}`);
  }

  handleDisconnect(client: Socket) {
    console.log(`Client disconnected: ${client.id}`);
  }

  // 클라이언트가 'message' 이벤트를 통해 메시지를 전송할 때 호출
  @SubscribeMessage('message')
  handleMessage(@MessageBody() data: { nickname: string, message: string }): void
{
    console.log("send ! : ", data)
    // 모든 클라이언트에게 메시지를 전송
    this.server.emit('message', { nickname: data.nickname, message: data.message
});
  }

  // 아래부터 치지직 채팅 소켓

  // 특정 이벤트 처리 (실시간 채팅 데이터 요청)
  @SubscribeMessage('requestChatData')
  handleRequestChatData(client: Socket, data: { streamerName: string }) {
    const { streamerName } = data;
    this.logger.log(`Chat data requested for streamer: ${streamerName}`);

    // 클라이언트와 스트리머 매핑 저장
```

```
    this.clientToStreamerMap.set(client.id, streamerName);

    // 이미 생성된 ChzzkChatService가 있으면 재사용
    // if (!this.chatServices.has(streamerName)) {
    if (true) {
      this.logger.log(`신규 추가 streamer: ${streamerName}`);
      const chzzkChatService = new ChzzkChatService(streamerName);
      this.chatServices.set(streamerName, chzzkChatService);

      // 새로운 메시지 발생 시 클라이언트에 전송
      chzzkChatService.on('newMessage', (message) => {
        this.server.to(client.id).emit('receiveChatData', { chatData: message });
      });

      // 채팅채널이 없는경우
      chzzkChatService.on('NoChatChannelId', (streamerStatus :object) => {
        this.server.to(client.id).emit('NoChatChannelId', { streamerName:
streamerName, streamerStatus });
      });
    }

    // 기존 메시지 반환
    const service = this.chatServices.get(streamerName);
    if (service) {
      client.emit('receiveChatData', { chatData: service.getChatMessages() });
    }

    // 클라이언트의 disconnect 이벤트 처리
    client.on('disconnect', () => {
      this.logger.log(`Client disconnected! : ${client.id}`);

      // 매핑에서 스트리머 이름 가져오기
      const disconnectedStreamer = this.clientToStreamerMap.get(client.id);

      if (disconnectedStreamer) {
        this.logger.log(`Cleaning up for streamer: ${disconnectedStreamer}`);

        // 리소스 정리
        const service = this.chatServices.get(disconnectedStreamer);
        if (service) {
          service.removeAllListeners(); // 등록된 이벤트 제거
          this.chatServices.delete(disconnectedStreamer); // 서비스클래스에서 제거
          service.onModuleDestroy();
          this.logger.log(`Service for streamer ${disconnectedStreamer} has been
cleaned up.`);
        }

        // 매핑 정보 삭제
        this.clientToStreamerMap.delete(client.id);
      }
    });
  }
}
```

- handleConnection: 클라이언트 연결 이벤트를 처리합니다.
- handleDisconnect: 클라이언트 연결 종료 이벤트를 처리합니다.
- handleMessage: 클라이언트에서 수신된 메시지를 모든 클라이언트로 브로드캐스트합니다.
- SubscribeMessage: 치지직 API 요청으로 실시간 채팅 데이터를 가져옵니다.

2 Twitch API 연동

🗁 **파일:** `twitch.service.ts` Twitch API를 통해 스트리머 정보를 가져오는 기능을 구현했습니다. NestJS의 HttpModule을 사용해 외부 API를 호출합니다.

```
import { Injectable } from '@nestjs/common';
import { HttpService } from '@nestjs/axios';
import { lastValueFrom } from 'rxjs'; // RxJS를 사용하여 HTTP 요청 결과 처리

@Injectable()
export class TwitchService {
  private clientId = process.env.TWITCH_CLIENT_ID; // 발급받은 Client ID
  private clientSecret = process.env.TWITCH_CLIENT_SECRET; // 발급받은 Client
Secret
  private accessToken: string | null = null;
  private readonly redirectUri = process.env.TWITCH_REDIRECT_URI

  constructor(private readonly httpService: HttpService) {}

  async getAccessToken(): Promise<string> {
    if (!this.accessToken) {
      // Access Token 요청
      const response$ = this.httpService.post(
        'https://id.twitch.tv/oauth2/token',
        null,
        {
          params: {
            client_id: this.clientId,
            client_secret: this.clientSecret,
            grant_type: 'client_credentials',
            redirect_uri: this.redirectUri, // 리디렉션 URL 추가
          },
        },
      );

      const response = await lastValueFrom(response$);
      this.accessToken = response.data.access_token;
    }
    return this.accessToken;
  }

  async getStreamerInfo(streamerNames: string[]) {
    const accessToken = await this.getAccessToken();
  // Streamer 이름들을 쉼표로 연결하여 쿼리로 추가
  const params = streamerNames.map(name => `user_login=${name}`).join('&');
```

```
    const response$ = this.httpService.get(
      `https://api.twitch.tv/helix/streams?${params}`,
      {
        headers: {
          Authorization: `Bearer ${accessToken}`,
          'Client-ID': this.clientId,
        },
      },
    );

    const response = await lastValueFrom(response$);
    return response.data.data; // 스트리머 정보 배열 반환
    }
  }
```

- getAccessToken: Twitch API에서 Access Token을 가져옵니다.
- getStreamerInfo: 스트리머 정보를 Twitch API를 통해 가져옵니다.
- HttpService: 외부 API 호출에 사용합니다.

3 치지직 API 연동

🗁 **파일:** `chzzkAPI.ts` 치지직(Chzzk) API를 통해 스트리머의 라이브 정보를 가져오고 채팅을 처리합니다. `ChzzkChatService` 클래스는 치지직 API를 사용해 스트리머의 라이브 방송 및 채팅 데이터를 가져오도록 설계되었습니다.

```
import { Injectable, Logger } from '@nestjs/common';
import * as ChzzkClient from 'chzzk';
import { EventEmitter } from 'events';

@Injectable()
export class ChzzkChatService extends EventEmitter {
  private readonly logger = new Logger('ChzzkChatService');
  private client: any;
  private chzzkChat: any;

  // 실시간 채팅 메시지를 저장하는 변수
  private chatMessages: string[] = [];

  constructor(streamerName: string) {
    super();
    this.initializeClient(streamerName);
  }

  // 치지직 클라이언트 초기화
  private async initializeClient(streamerName: string) {
    this.client = new ChzzkClient.ChzzkClient();
    const channelName = streamerName;
    const result = await this.client.search.channels(channelName);
    const channel = result.channels[0];

    if (!channel) {
```

```
        this.logger.error(`Channel "${channelName}" not found.`);
        return;
      }

      const liveDetail = await this.client.live.detail(channel.channelId);

      if (liveDetail) {
        const media = liveDetail.livePlayback?.media ? liveDetail.livePlayback.media
: null;
        if (media) {
          const hls = media.find((media) => media.mediaId === 'HLS');

          if (hls) {
            const m3u8 = await this.client.fetch(hls.path).then((r) => r.text());
            this.logger.log(`HLS Playlist: ${m3u8}`);
          }
        }
      }

      // 채팅 연결
      this.connectChat(channel.channelId);
    }

    // 채팅 연결 및 이벤트 설정
    private connectChat(channelId: string) {
      this.chzzkChat = this.client.chat({
        channelId,
        pollInterval: 30 * 1000, // 30초마다 변경 감지
      });

      this.chzzkChat.on('connect', () => {
        this.logger.log('Connected to Chzzk Chat');
        this.chzzkChat.requestRecentChat(50); // 최근 50개 채팅 요청
      });

      this.chzzkChat.on('chat', (chat: any) => {
        const message = chat.hidden ? '[블라인드 처리 됨]' : chat.message;
        const nickname = chat.profile.nickname;

        // 메시지를 저장
        this.chatMessages.push(`${nickname}: ${message}`);
        this.emit('newMessage', { nickname, message });
      });

      this.chzzkChat.on('systemMessage', (systemMessage: any) => {
        this.logger.log(`System Message: ${systemMessage.extras.description}`);
      });

      this.chzzkChat.connect();
    }

    // 외부에서 실시간 메시지 접근
    public getChatMessages(): string[] {
      return this.chatMessages;
```

```
    }

    // 서비스 종료 시 자원 정리
    async disconnect() {
      if (this.chzzkChat) {
        await this.chzzkChat.disconnect();
        this.logger.log('Disconnected from Chzzk Chat');
      }
    }
}
```

- 클래스 기반 설계: ChzzkChatService는 스트리머 이름을 기반으로 치지직 클라이언트를 초기화하고 채팅 데이터를 관리합니다.
- 라이브 정보 가져오기: 스트리머의 HLS 미디어 URL을 가져와 로그로 출력합니다.
- 채팅 데이터 수신: 채팅 메시지, 시스템 메시지 등의 데이터를 수신하여 저장하거나 이벤트를 발생시킵니다.
- 이벤트 기반 처리: EventEmitter를 사용하여 새로운 채팅 메시지가 발생할 때 이를 외부에서 처리할 수 있도록 설계되었습니다.

⚙ 서비스 연동 방법

🗁 **파일:** `chat.gateway.ts` ChzzkChatService는 특정 스트리머의 채널 데이터를 가져오고 채팅 데이터를 처리합니다. 아래는 ChatGateway에서 이 서비스를 사용하는 예시입니다.

```
@SubscribeMessage('requestChatData')
handleRequestChatData(client: Socket, data: { streamerName: string }) {
  const { streamerName } = data;
  const chzzkChatService = new ChzzkChatService(streamerName);

  chzzkChatService.on('newMessage', (message) => {
    client.emit('receiveChatData', message); // 클라이언트에게 실시간 채팅 데이터 전
송
  });

  // 기존 메시지 반환
  client.emit('receiveChatData', { chatData: chzzkChatService.getChatMessages()
});
}
```

- handleRequestChatData: 스트리머의 채팅 데이터를 요청받아 처리합니다.
- newMessage 이벤트: 실시간 채팅 메시지를 수신하여 해당 클라이언트로 전송합니다.
- getChatMessages: 기존 채팅 데이터를 반환합니다.

✿ 주요 화면 설명

메인 페이지

- 스트리머 목록과 플랫폼 필터 버튼 제공.

스트리밍 페이지.

- Twitch, YouTube, 치지직 영상 스트리밍 + 실시간 채팅 제공.

🔗 GitHub Repository

- 프로젝트의 모든 코드는 GitHub에서 확인할 수 있습니다:
- 👉 https://github.com/DaeSoeps/zhzzk

🛠 설치 및 실행 방법

레포지토리 클론

- git clone https://github.com/DaeSoeps/zhzzk.git
- cd zhzzk

의존성 설치

- npm install

환경 변수 설정

- .env.local 파일 생성 후 아래와 같이 설정:
- NEXT_PUBLIC_BACK_URL=https://zhzzk-back.onrender.com

개발 서버 실행

- npm run dev

💡 배운 점 및 개선 방향

배운 점

- 프론트엔드 최신 기술 학습: 기존 React, Next 버전과의 차이점과 신버전에 학습
  - **Next.js 15 이상**: `page Route`와 `Dynamic Routing` 등 최신 기능의 도입 및 활용.
  - **React 최신 패턴**: React 18 이상에서의 변경점.
  - **Tailwind CSS**: CSS-in-JS 대신 유틸리티 기반 CSS를 활용하여 생산성과 유지보수성을 모두 확보.
  - **Zustand 상태 관리**: Redux 대체 라이브러리로 최신 상태 관리 트렌드를 경험.
- API 연동: 치지직, 트위치를 비롯한 서드파티 라이브러리 활용 및 데이터 처리 방식 학습.
- WebRTC 활용: 화면 공유 및 스트리밍 기능 구현.
- Firebase 연동: 클라우드 기반 데이터 관리.
- 모듈화된 서비스 설계: NestJS 모듈 구조로 서비스의 독립성을 유지.
- EventEmitter 활용: 이벤트 기반 설계를 통해 실시간 데이터 처리를 구현.
- CI/CD: Azure DevOps를 통한 자동 배포 경험.

개선 방향

- 성능 최적화: 불필요한 요청 및 렌더링 최소화.
- UX 개선: 모바일 환경에서의 레이아웃 최적화.

- 로그 관리: 더 나은 에러 추적 및 로그 시스템 구축.
- 모니터링 도구 도입: (Prometheus, Grafana 등) 실시간 서비스 상태 모니터링 추가.