Pusan National University
Computer Science and Engineering
Technical Report 2025-09

AR 기반 실시간 원격 협업 시스템 구현

202055569 윤민혁

202055507 강유승

202055525 김준성

지도교수 이명호

목 차

1. 서론

1.1. ..... 연구 배경 및 기존 문제점

1.2. ..... 연구 목표

2. 연구 배경

2.1. ..... AR

2.2. ..... WebRTC

3. 연구 내용

3.1. ..... 개발 환경 비교

3.2. ..... 데이터베이스

3.3. ..... AR 환경 비교

3.4. ..... 화상통화 환경 구현

3.5. ..... AR 구현

3.6. ..... AWS

3.7. ..... Node.js + socket.io

3.8. ..... 프로젝트 기능 설명

4. 연구 결과 분석 및 평가

4.1. ..... ARSOL 프로젝트 보완 및 대응 방안 계획

4.2. ..... 개발 일정

4.3. ..... 역할 분담

4.4. ..... 시스템 구성도

4.5. ..... API 명세서

4.6. ..... 협업 방식

5. 결론 및 향후 연구 방향

5.1. ..... 결론 및 기대효과

5.2. ..... 향후 연구 방향

6. 참고 문헌

1. 서론

1.1. 연구 배경 및 기존 문제점

컴퓨터 비전문가는 PC 조립에 어려움을 겪으며, 기존 설명서나 영상으로는 개인 맞춤형 도움을 받기가 어렵다.

사용자가 보유한 실제 부품을 기반으로 한 맞춤형 안내의 필요성이 대두된다.

1.2. 연구 목표

본 연구는 AR(증강현실) 기술과 실시간 원격 통신 기술을 결합하여, 전문 지식이 없는 사용자도 PC 조립 및 부품 교체 과정에서 전문가의 실시간 지원을 직관적으로 받을 수 있는 원격 협업 시스템을 구현하는 것을 목표로 한다. 이를 통해 달성하고자 하는 세부 목표는 다음과 같다.

AR 기반의 직관적인 맞춤형 가이드 제공: 사용자의 실제 부품 위에 3D 객체나 디지털 주석을 증강하여, 복잡한 조립 과정을 시각적으로 이해하기 쉽게 안내한다.

실시간 양방향 소통을 통한 문제 해결: WebRTC 기반의 화상 통화와 데이터 채널을 통해 사용자와 전문가가 원활하게 소통하며 조립 과정에서 발생하는 돌발 상황에 즉각적으로 대처할 수 있도록 지원한다.

웹 기반 솔루션으로의 접근성 확보: 별도의 프로그램 설치 없이 웹 브라우저만으로 서비스에 접근하게 하여, 사용자의 편의성을 극대화하고 서비스 이용의 진입 장벽을 낮춘다.

2. 연구 배경

2.1. AR

AR(Augmented Reality, 증강현실) 기술을 도입한 이유는 사용자에게 직관적이고 몰입감 높은 원격 협업 경험을 제공하기 위함이다. 특히 별도의 애플리케이션 설치 없이 웹 브라우저를 통해 즉시 접근 가능한 Web AR을 핵심 기술로 채택했으며, 그 이유는 다음과 같다.

1. 뛰어난 접근성

사용자가 앱 스토어에서 별도의 애플리케이션을 다운로드하고 설치할 필요 없이, 웹사이트 주소(URL)에 접속하는 것만으로 즉시 AR 기능을 사용할 수 있다. 이는 서비스 이용의 진입 장벽을 크게 낮추어 사용자의 참여를 유도하는 데 결정적인 장점으로 작용한다.

2. 크로스 플랫폼 호환성

Web AR은 단일 코드로 개발하여 iOS, Android, 데스크톱 등 주요 운영체제의 최신 웹 브라우저에서 동일하게 동작한다. 이를 통해 플랫폼별로 앱을 따로 개발하고 유지보수해야 하는 부담을 줄이고, 개발 및 배포 효율성을 극대화할 수 있다.

3. WebRTC와의 완벽한 통합

Web AR은 웹 표준 기술을 기반으로 하므로, 본 프로젝트의 또 다른 핵심 기능인 WebRTC와의 결합이 자연스럽다. 이를 통해 원격지의 사용자와 화상으로 소통하면서, 동일한 현실 공간의 특정 사물(이미지 타겟) 위에 가상의 객체나 주석을 함께 띄워놓고 상호작용하는 'See-What-I-See' 형태의 고도화된 원격 협업 시나리오 구현이 가능하다.

4. 신속한 개발 및 배포

웹 기반 서비스는 네이티브 앱에 비해 개발 주기가 짧고, 기능 수정이나 업데이트 시 사용자가 별도로 앱을 업데이트할 필요 없이 서버 배포만으로 모든 사용자에게 즉시 새로운 버전을 제공할 수 있다. 이는 빠르게 변화하는 요구사항에 민첩하게 대응할 수 있게 한다.

5. 안정적인 마커 기반 AR

프로젝트에서는 특정 이미지를 인식하여 콘텐츠를 증강하는 '마커(Marker) 기반' 방식을 사용한다. 이 방식은 특정 사물이나 문서에 디지털 정보를 정확하게 고정시켜야 하는 원격 교육, 비대면 유지보수, 제품 설명 등의 시나리오에서 매우 안정적이고 효과적인 사용자 경험을 제공한다.

이러한 장점들을 바탕으로, Web AR은 복잡한 설치 과정 없이 누구나 쉽게 접근하여 실시간으로 소통하고 상호작용할 수 있는 AR 협업 솔루션(ARsol)을 구축하기 위한 최

적의 기술로 판단되어 도입하게 되었다.

2.2. WebRTC

화상통화 구현에 WebRTC를 사용한 이유는 브라우저에서 별도의 프로그램 설치 없이 카메라와 마이크에 접근할 수 있기 때문이며, P2P 연결을 통해 영상과 음성이 서버를 거치지 않고 직접 전달되어 지연 시간이 낮고 효율적이기 때문이다.

또한 주요 브라우저에서 기본적으로 지원하는 표준 기술이기 때문에 호환성과 안정성이 뛰어나며, STUN과 TURN을 통한 NAT 우회 기능을 제공하여 다양한 네트워크 환경에서도 원활한 연결이 가능하다.

더불어 음성과 영상뿐만 아니라 데이터 채널까지 지원하여 채팅이나 파일 전송 같은 부가 기능 확장이 가능하며, 기본적으로 암호화가 적용되어 안전한 통신이 보장되기 때문에 화상통화 구현에 WebRTC를 사용하게 되었다.

3. 연구 내용

3.1. 개발 환경 비교

개발 중간에 Unity -> React + Next.js로 개발 환경을 변경

Unity	핵심 기능을 제외한 대부분의 기능을 구현하는 데에 문제가 없다는 장점이 있다. 그러나, WebRTC 패키지의 버전 및 AR과의 호환 문제, 너무 긴 지연 시간으로 인해, 서비스 제공에 무리가 있다고 판단하여 개발 환경을 교체하게 되었다.
React + Next.js	React 개발 경험은 상대적으로 부족했으나, 다양한 패키지를 지원하며, 참고할 문서들이 많기 때문에 핵심 기능들을 구현하는 데에 큰 무리가 없다고 판단하였다. 배포 및 서버 관리도 Unity에 비해 어렵지 않다는 장점이 있다 . 실제로 계획했던 모든 기능을 구현하는데 성공하였으며, 배포도 안정적으로 이루어지고 있다

3.2. 데이터베이스

3.2.1. 데이터베이스 비교

Firebase

1. 개요: Firebase는 Google이 운영하는 NoSQL 기반 BaaS로, Firestore와 Realtime Database를 통한 유연한 데이터 구조, 실시간 동기화를 지원한다.

2. 장점: Firebase는 대규모 서비스에서도 안정성이 높으며 실시간 동기화와 인증 시스템과 같은 원스톱 서비스 제공에서도 우수한 성능을 보인다.

3. 단점: Firebase의 Storage 기능은 전면 유료화 된 상태로, 미디어 파일을 저장하기에는 비용 부담 발생의 문제점이 존재한다.

Supabase

1. 개요: Supabase는 PostgreSQL 기반의 오픈소스 BaaS로, 관계형 데이터, 고급 SQL 쿼리, Row Level Security 등 전통적인 SQL DB의 기능을 지원한다.

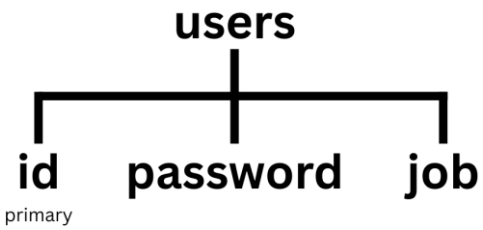
2. 장점: Supabase는 복잡한 SQL 쿼리, 오픈소스 및 직접 제어에 용이하며, Storage 기능을 무료로 제공하여 미디어 파일들을 별다른 비용 없이 관리할 수 있다.

3. 단점: Supabase는 신생 서비스인 이유로 문서와 한글 자료가 부족하여 시스템을 구축하는데 어려움이 있을 수 있다.

위 데이터베이스들을 비교한 결과, 회원 정보, 전문가 리뷰 정보의 데이터는 실시간 동기화에 용이한 Firebase에 저장, 통화 화면 녹화 영상 같은 미디어 정보 데이터는 비용 부담이 없는 Supabase에 저장하도록 하였다.

3.2.2. 구현 내용

a. 회원 정보



- 클라이언트가 회원가입을 통해 입력된 회원 정보(아이디, 비밀번호, 역할)는 위 구조로 Firebase DB에 저장된다.

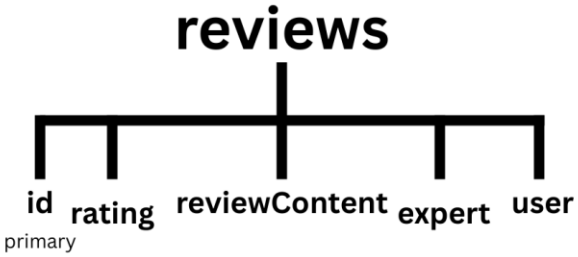
- 클라이언트가 로그인할 때는 입력한 아이디와 비밀번호를 DB에 저장된 값과 비교하는데, 이 과정에서 비밀번호는 동일한 해싱 과정을 거쳐 매칭을 시도하여 로그인 성공 여부를 판별한다.

- 세션 선택 페이지에서 클라이언트의 역할이 '사용자'일 경우에만 세션 생성이 가능하다.

- 화상 통화 세션 종료 시, 클라이언트의 역할이 '사용자'인 경우 곧바로 리뷰 작성 페이지로 이동하여 전문가에 대한 후기 및 별점을 남길 수 있다.

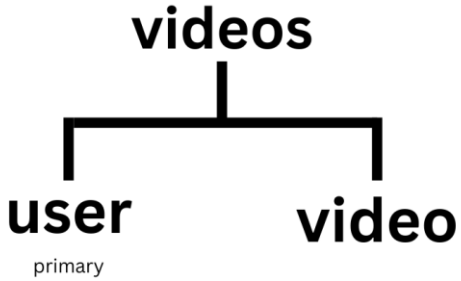
- 리뷰 조회 페이지에서는, 로그인한 계정의 역할에 따라 볼 수 있는 정보가 달라진다. 사용자인 경우 본인이 남긴 모든 리뷰 목록을 확인할 수 있고, 전문가의 경우 본인에 대해 사용자가 남긴 피드백과 평점을 열람할 수 있다.

b. 전문가 리뷰 정보



- 사용자가 전문가에게 작성한 리뷰 정보는 위 구조로 Firebase DB에 저장된다
- 각 리뷰의 id 값은 사용자가 생성한 세션의 roomId로 설정되어, 특정 세션에 대한 리뷰임을 명확히 구분한다.
- 리뷰에는 별점(rating)이 1~5까지의 정수로 포함되며, 이 값에 따라 별표 개수 등 시각적 표시가 자동으로 이루어진다.
- 리뷰 확인 페이지에서 클라이언트가 사용자라면 user 필드가 본인의 id와 일치하는 리뷰를, 전문가라면 expert 필드가 본인 id인 리뷰를 각각 확인할 수 있다.

c. 통화 화면 녹화 영상



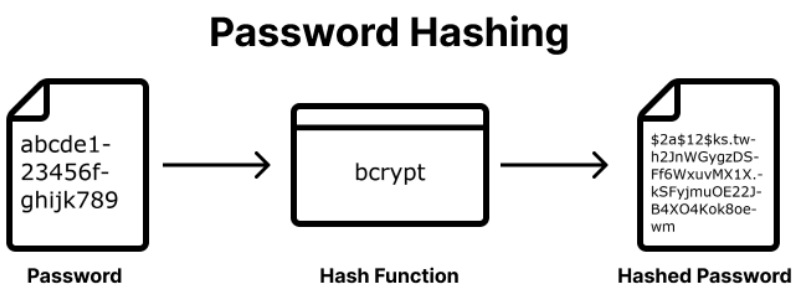
- 클라이언트가 녹화한 영상은 webm 형식으로 Supabase DB에 저장된다.
- 사용자가 녹화 영상 조회 기능을 이용하면, 시스템은 Supabase에 저장된 영상 중 영상의 user 값과 현재 클라이언트의 id 값이 일치하는 파일만을 불러와 보여준다. 이를 통해 사용자는 본인이 직접 녹화한 세션 영상만을 신속하게 찾고 시청할 수 있다.

- 녹화 영상의 파일명은 YYYYMMDDHHMM.webm 형식으로 저장되어 녹화가 이루어진 정확한 날짜와 시간을 식별할 수 있다.

3.2.3. 회원 비밀번호 암호화

회원가입 시 입력한 비밀번호는 bcrypt 라이브러리(bcryptjs)의 hashSync 함수로 해싱되어 DB에 저장되며, 이후 로그인 시 입력한 비밀번호를 해싱하여 DB에 저장된 해시값을 비교하여 일치 여부를 판단한다.

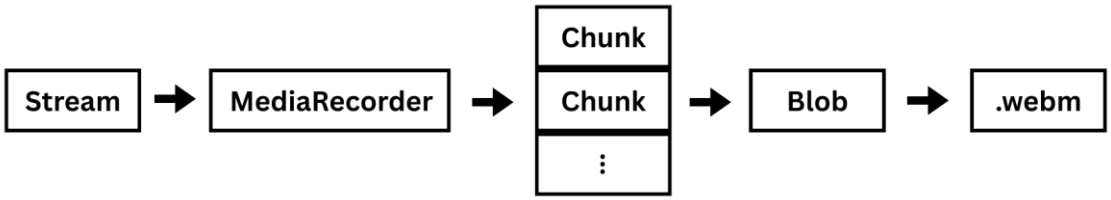
bcrypt는 고유한 salt를 비밀번호에 추가한 후 연산을 반복하여 해시값을 생성하는데 동일한 비밀번호라도 salt값이 모두 다르기 때문에 해시 결과는 완전히 달라진다. 그러므로 DB 가 유출되더라도 해시값만을 저장하므로 탈취자는 원래의 비밀번호를 알기 어렵게 되는 이점을 가진다.



3.2.4. 화면 녹화

화면 녹화는 웹 표준의 MediaRecorder API를 사용하였다.
MediaRecorder는 녹화된 Chunk를 순차적으로 받아 배열에 저장한 후, 녹화 종료 시 Blob을 생성하여 하나의 webm 파일로 변환 후 Supabase DB에 저장한다.

MediaRecorder는 별도 플러그인이나 외부 소프트웨어 없이 객체 하나만으로 녹화와 저장이 가능하며, 실시간 인코딩을 통해 대용량 미디어 데이터도 효과적으로 다룰 수 있는 이점을 지닌다.



3.3. AR 환경 비교

본 프로젝트는 웹 기반 증강현실(AR) 구현을 위해 마커리스(Marker-less) 방식의 이미지 트래킹을 지원하는 MindAR 라이브러리를 채택하였다.

MindAR는 A-Frame 및 Three.js와 통합되어 웹 환경에서 효율적으로 3D 물체를 렌더링하고 사용자 상호작용을 처리할 수 있는 장점이 있다. 이는 별도의 애플리케이션 설치 없이 웹 브라우저만으로 AR 콘텐츠에 접근할 수 있게 하여 사용자 접근성을 극대화한다.

3.4. 화상통화 환경 구현

3.4.1. 작동 원리

브라우저가 HTTPS 환경에서 getUserMedia를 호출해 카메라와 마이크 권한을 요청한다. 권한이 승인되면 MediaStream 트랙(video/audio)을 확보한다. 화면 공유가 필요하면 getDisplayMedia로 별도의 트랙을 확보한다.

방에 입장하면 클라이언트가 socket.io로 시그널링 서버에 접속한다. 방 아이디로 join 이벤트를 보내고, 같은 방의 기존 참가자 목록을 수신한다.

피어 연결 객체를 생성한다. 각 피어마다 RTCPeerConnection을 생성하고, STUN/TURN 서버 설정(iceServers)을 주입한다. STUN으로 공인 후보를 수집하고, TURN은 최후 수단 릴레이용으로 사용한다. TURN이 미구성된 경우 대칭 NAT 환경에서는 실패 가능성이 높다.

트랙을 추가한다. 로컬 MediaStream의 audio/video 트랙을 pc.addTrack으로 RTCPeerConnection에 붙인다. 데이터 채널이 필요할 경우 createDataChannel로 미리 생성한다.

오퍼와 앤서를 교환한다. 새 참가자가 기존 참가자에게 SDP offer를 생성(pc.createOffer → setLocalDescription) 후 socket.io로 전송한다. 상대는 setRemoteDescription → createAnswer → setLocalDescription → answer를 다시 시그널링으로 회신한다. 양쪽 모두 최종적으로 setRemoteDescription까지 완료하면 코덱, 암호화, 전송 파라미터가 합의된다.

ICE 후보를 수집하고 교환한다. pc.onicecandidate에서 로컬 ICE 후보가 발생하면 시그널링으로 전송한다. 상대는 addIceCandidate로 반영한다. 후보가 직접 연결 가능하면 P2P 경로가 확립된다. 불가할 경우 TURN 릴레이로 전환된다.

미디어를 수신하고 표시한다. pc.ontrack에서 원격 트랙을 수신한다. React 컴포넌트에서 `<video>` 요소의 srcObject에 MediaStream을 바인딩해 화면과 오디오를 재생한다. 여러 명이 참가한 경우 참가자별 비디오 요소를 동적으로 렌더링한다.

연결 상태를 관리한다. pc.iceconnectionstatechange와 pc.connectionstatechange 이벤트로 연결 상태를 모니터링한다. disconnected나 failed가 감지되면 재시도 로직을 수행한다(ICE restart: createOffer({iceRestart:true}) → 재협상). 탭이 백그라운드로 전환되거나

모바일 네트워크가 바뀔 때 자주 발생한다.

재협상 및 트랙 변경을 수행한다. 화면 공유 시작과 종료, 마이크 on/off, 카메라 전환 시 트랙을 교체한다(pc.getSenders()[i].replaceTrack). 코덱이나 해상도 변경이 필요하면 renegotiationneeded 이벤트에 따라 새 오퍼를 생성해 교환한다.

시그널링 서버는 오직 메시지 중계(SDP, ICE, 방 입퇴장 이벤트)만 담당한다. 미디어 패킷은 서버를 통과하지 않는다(P2P). 다자간에서는 메시지 팬아웃만 증가한다. 서버 과부하는 주로 접속이나 퇴장 스파이크 때 발생한다.

nginx는 80/443을 수신하고 443에서 TLS를 종료한다(Let's Encrypt 인증서). 웹앱 정적 파일을 서빙하거나 Node.js 프런트엔드로 프록시한다. socket.io의 WebSocket 업그레이드 헤더를 전달한다(Upgrade/Connection 유지). HTTPS 제공으로 브라우저 보안 컨텍스트 요건을 충족한다(getUserMedia와 RTCPeerConnection 필수).

pm2는 시그널링 서버(Node.js) 프로세스를 실행, 재시작, 로그 관리한다. 서버 장애나 배포 시 무중단에 가까운 재시작(pm2 reload)이 가능하다.

네트워크와 방화벽에서는 EC2 보안 그룹에서 80/443을 오픈한다. 시그널링 포트(예: 3000)는 내부만 열고 외부는 nginx로만 진입시키는 구성이 권장된다. TURN을 사용할 경우 3478(UDP/TCP), 5349(TLS), 릴레이 포트 범위를 열어야 한다.

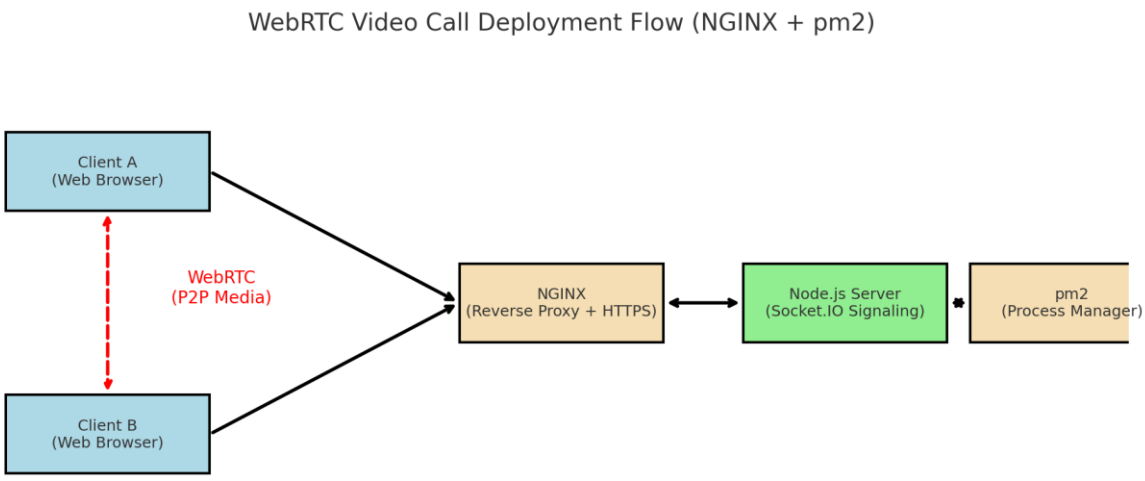
성능과 품질은 해상도와 프레임레이트 제약을 getUserMedia constraints로 조절한다(예: 1280×720, 30fps). 대역폭 제한은 sender.setParameters(encodings)로 적용 가능하다(최대 비트레이트). 네트워크 약화 시 WebRTC가 자체적으로 적응(콘제스천 컨트롤)을 수행한다.

보안과 프라이버시는 HTTPS를 강제하고, 시그널링 토큰으로 방 접근을 제어하며, 임의의 방 아이디 추측을 방지해야 한다. 민감 권한은 사용자 제스처 기반으로만 요청한다.

모니터링과 디버깅은 getStats로 비트레이트, 패킷 손실, RTT를 수집해 UI에 노출할 수 있다. 시그널링 로그는 pm2 logs로 확인한다.

정리하면, 시그널링(socket.io)이 초기 협상과 후보 교환을 중계하고, RTCPeerConnection이 STUN/TURN을 통해 최적 경로를 찾아 P2P로 미디어를 교환한다. nginx가 도메인, HTTPS, 프록시를 제공해 브라우저 보안 요건과 배포 편의성을 해

결하고, pm2가 시그널링 서버의 가용성을 보장한다.



3.4.2. 구현 방법

시그널링 서버는 브라우저끼리 직접 연결하기 전에 필요한 정보를 교환해주는 역할을 한다. 구체적으로는 SDP(Offer/Answer)와 ICE 후보를 교환하도록 중계한다.

```
import express from "express";
import http from "http";
import { Server } from "socket.io";

const app = express();
const server = http.createServer(app);
const io = new Server(server, {
  cors: { origin: "*" },
});
io.on("connection", (socket) => {
  console.log("client connected:", socket.id);

  socket.on("join", (roomId) => {
    socket.join(roomId);
    socket.to(roomId).emit("new-peer", socket.id);
  });
  socket.on("offer", (data) => {
    socket.to(data.target).emit("offer", { sdp: data.sdp, from: socket.id });
  });
  socket.on("answer", (data) => {
    socket.to(data.target).emit("answer", { sdp: data.sdp, from: socket.id });
  });
  socket.on("ice-candidate", (data) => {
    socket.to(data.target).emit("ice-candidate", { candidate: data.candidate, from: socket.id });
  });
  socket.on("disconnect", () => {
    io.emit("peer-disconnected", socket.id);
  });
});

```

브라우저는 카메라와 마이크 스트림을 가져와서 `RTCPeerConnection`을 생성하고, 시그널링 서버를 통해 Offer/Answer/ICE를 교환한 뒤 P2P 연결을 맺는다.

```jsx
import React, { useEffect, useRef } from "react";
import { io } from "socket.io-client";

const socket = io("https://ggg-ar-sol.org", { transports: ["websocket"] });

function App() {
  const localVideo = useRef(null);
  const remoteVideo = useRef(null);
  const pcRef = useRef(null);
  const roomId = "test-room";

  useEffect(() => {
    async function init() {
      const stream = await navigator.mediaDevices.getUserMedia({ video: true, audio: true });
      localVideo.current.srcObject = stream;
      pcRef.current = new RTCPeerConnection({
        iceServers: [{ urls: "stun:stun.l.google.com:19302" }],
      });
      stream.getTracks().forEach((track) => pcRef.current.addTrack(track, stream));
      pcRef.current.ontrack = (e) => {
        remoteVideo.current.srcObject = e.streams[0];
      };
      pcRef.current.onicecandidate = (e) => {
        if (e.candidate) {
          socket.emit("ice-candidate", { target: "all", candidate: e.candidate });
        }
      };
      socket.emit("join", roomId);
    }

    init();
    socket.on("new-peer", async (peerId) => {
      const offer = await pcRef.current.createOffer();
      await pcRef.current.setLocalDescription(offer);
      socket.emit("offer", { target: peerId, sdp: offer });
    });
    socket.on("offer", async ({ sdp, from }) => {
      await pcRef.current.setRemoteDescription(new RTCSessionDescription(sdp));
      const answer = await pcRef.current.createAnswer();
      await pcRef.current.setLocalDescription(answer);
      socket.emit("answer", { target: from, sdp: answer });
    });
    socket.on("answer", async ({ sdp }) => {
      await pcRef.current.setRemoteDescription(new RTCSessionDescription(sdp));
    });
    socket.on("ice-candidate", async ({ candidate }) => {
      try {
        await pcRef.current.addIceCandidate(new RTCIceCandidate(candidate));
      } catch (err) {
        console.error("Error adding ICE candidate", err);
      }
    });
  }, []);

  return (
    <div>
      <video ref={localVideo} autoPlay playsInline muted style={{ width: "300px" }} />
      <video ref={remoteVideo} autoPlay playsInline style={{ width: "300px" }} />
    </div>
  );
}

export default App;
```

NGINX는 도메인과 HTTPS를 처리하고, React 앱과 Node.js 시그널링 서버로 요청을 프록시한다.

```
server {
    listen 80;
    server_name ggg-ar-sol.org www.ggg-ar-sol.org;

    location / {
        proxy_pass http://localhost:3000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";
        proxy_set_header Host $host;
    }

    location /socket.io/ {
        proxy_pass http://localhost:4000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";
        proxy_set_header Host $host;
    }
}
```

Node.js 시그널링 서버를 pm2로 실행하면 꺼지지 않고 계속 돌릴 수 있다.

```
pm2 start server.js --name signaling
pm2 startup
pm2 save
```

3.5. AR 구현

3.5.1. 작동 원리

1. 이미지 인식

시스템은 먼저 기기의 카메라를 통해 실시간으로 주변 환경을 비춘다. 그리고 미리 등록된 특정 이미지('이미지 타겟')를 현실 공간에서 찾아낸다.

2. 좌표계 정합

카메라가 이미지 타겟을 성공적으로 인식하면, 시스템은 그 이미지의 위치와 방향을 기준으로 3D 가상 공간의 좌표계를 생성하고 일치시킨다. 즉, 이미지 타겟이 가상 공간의 원점(0,0,0)이 된다.

3. 콘텐츠 증강

이 가상 공간 위에 3D 모델(CPU, RAM 등), 텍스트, 도형 같은 디지털 콘텐츠를 배치한다. 이 콘텐츠들은 이미지 타겟 좌표계를 기준으로 위치하기 때문에, 사용자가 카메라를 움직여도 마치 현실의 이미지 위에 고정되어 있는 것처럼 보인다.

4. 실시간 렌더링

시스템은 매 순간 (1) 카메라를 통해 들어오는 현실 영상과 (2) 이미지 타겟 위에 증강된 가상 3D 콘텐츠를 하나로 합성하여 화면에 보여준다. 이 과정이 매우 빠르게 반복되어 사용자는 끊김 없는 증강현실을 경험하게 된다.

5. 다중 사용자 상호작용

여러 사용자가 같은 AR 세션에 참여할 경우, 한 사용자가 가상 객체를 추가하거나 그림을 그리는 등의 행동을 하면 그 행동에 대한 정보(예: "A 위치에 CPU 모델 추가")가 서버를 통해 다른 모든 사용자에게 즉시 전송된다. 정보를 수신한 다른 사용자들의 기기에서도 동일한 행동이 재현되어, 모든 참여자가 같은 AR 장면을 실시간으로 보고 상호작용할 수 있게 된다.

3.5.2. 구현 방법

1. 핵심 라이브러리 활용

- **MindAR.js & A-Frame**: AR 기능의 기반이 되는 라이브러리다. HTML과 유사한 A-Frame 태그(<a-scene>, <a-entity>)를 사용하여 3D 장면을 선언적으로 구성한다. <a-scene>의 mindar-image 속성에 이미지 타겟 파일(targets.mind) 경로를 지정해 MindAR의 이미지 트래킹 기능을 활성화한다.

- **React**: 전체 AR 기능을 하나의 재사용 가능한 ARComponent로 캡슐화하고, useEffect 훅을 사용해 카메라 초기화, AR 엔진 시작과 정지 등 컴포넌트의 생명주기에 맞춘 작업들을 관리한다.

2. AR 장면 및 카메라 설정

useEffect 훅 내부에서 navigator.mediaDevices.getUserMedia API를 호출해 카메라 권한을 얻고 비디오 스트림을 가져온다. 가져온 비디오 스트림은 화면에 보이지 않는 <video> 태그에서 재생시키고, AR 콘텐츠가 렌더링되는 캔버스와 실제 비디오 화면을 requestAnimationFrame을 통해 매 프레임 combinedCanvasRef에 합쳐 그려 최종 화면을 구현한다.

3. 사용자 상호작용 처리

이미지 타겟이 인식되면 그 위에 나타나는 투명한 평면(class="target-plane")에 click 이벤트 리스너를 추가한다. 클릭 이벤트가 발생하면 handleClick 함수가 실행되어, 현재 선택된 도구(selectedTool)에 따라 document.createElement를 통해 동적으로 A-Frame 엘리먼트(a-cone, a-text 등)를 생성하고 장면에 추가한다. .gltf 3D 모델은 <a-entity>에 gltf-model 속성을 설정해 로드한다.

4. 실시간 협업 기능 구현

- **Socket.IO**: 서버와의 실시간 통신을 담당한다. 상위 컴포넌트로부터 socket 객체를 props로 전달받는다.
- **이벤트 수신**: 다른 사용자의 행동(객체 추가, 그리기)에 대한 정보를 담은 props(peerClickCoords, drawData)의 변경을 useEffect 훅으로 감지한다. 변경이 감지되면 해당 정보를 바탕으로 로컬 AR 장면에 동일한 객체를 생성하거나 그림을 그린다.
- **실시간 그리기**: 특히 그리기 기능은 A-Frame만으로는 구현하기 어려워, 기반 라이브러리인 Three.js를 직접 사용한다. THREE.Line과 BufferGeometry를 생성하고, 실시간으로 수신되는 좌표를 지오메트리의 정점(vertex) 데이터에 계속

추가해 선이 그려지는 것을 구현한다.

3.6. AWS

EC2

서비스 배포를 위한 수단으로 AWS를 이용하기로 결정했다.
그중 EC2라는 IaaS(Infra as a Service)를 활용하였고, Elastic IP, pm2, nginx를 이용해 편리하게 서비스 관리 및 배포가 가능하도록 구현했다.

Elastic IP

Elastic IP는 AWS에서 제공하는 고정 퍼블릭 IP 주소를 의미한다.
일반적으로 EC2 인스턴스를 생성하면 퍼블릭 IP가 할당되지만, 인스턴스를 중지 후 재시작하면 그 IP는 바뀔 수 있다는 문제점이 있다.

반면 Elastic IP는 사용자가 직접 할당받아 특정 인스턴스에 연결하기 때문에 인스턴스를 껐다 켜도 IP가 변하지 않는다는 특징이 있다.
서비스는 고정된 IP가 필요하므로 Elastic IP를 사용하여 운영하기로 결정했다.

PM2

PM2는 Node.js 애플리케이션을 관리하는 프로세스 매니저다.
node app.js로 실행하면 터미널을 닫는 순간 애플리케이션이 종료되지만, PM2를 사용하면 애플리케이션이 백그라운드에서 계속 실행된다.
애플리케이션이 오류로 죽더라도 자동으로 다시 시작되고, 서버에 여러 CPU 코어가 있을 경우 클러스터 모드로 실행할 수 있다.

실행 로그를 모아 관리할 수 있고 메모리 사용량 같은 상태를 모니터링할 수 있다.
결국 PM2는 Node.js 애플리케이션을 단순 실행하는 것이 아니라 운영 환경에서 안정적으로 지속적으로 동작하게 해주는 도구다.

Nginx

nginx는 웹서버이자 리버스 프록시 서버다. 리버스 프록시는 클라이언트에서 들어온 요청을 받아 내부에서 돌아가는 다른 애플리케이션 서버(Node.js 등)로 전달해주는 역할을 한다. 배포 과정은 먼저 도메인 ggg-ar-sol.org를 구매하고, 그 도메인을 EC2 서

버의 공인 IP(Elastic IP)에 연결한다. 그다음 EC2 안에서 Node.js 애플리케이션을 pm2로 실행한다.

하지만 외부에서는 직접 3000번 포트 같은 내부 포트로 접근할 수 없으므로 nginx를 설치하고 설정 파일을 작성해 server_name ggg-ar-sol.org로 들어온 요청을 잡는다. nginx는 그 요청을 내부 포트에서 실행 중인 Node.js 서버로 전달한다. 동시에 nginx는 SSL 인증서를 붙일 수 있어서 https 연결을 제공한다.

결과적으로 사용자는 브라우저 주소창에 ggg-ar-sol.org만 입력해도 nginx가 중간에서 트래픽을 받아 내부 서버에 전달하고, 응답을 다시 클라이언트로 돌려주게 된다.

아래의 표는 배포를 하기 위한 보안 규칙 테이블이다.

프로토콜	포트	출처	설명
TCP	443	0.0.0.0/0	HTTPS 접속 허용
ICMP	전체	0.0.0.0/0	ICMP 허용
TCP	4000	0.0.0.0/0	Backend 포트
TCP	3000	0.0.0.0/0	Frontend 포트
TCP	22	0.0.0.0/0	SSH 접속 허용

3.7. Node.js + socket.io

이 프로젝트에서 Node.js는 시그널링 서버의 기반이 되는 런타임 환경으로 사용되었다. 브라우저 간의 화상통화 자체는 WebRTC가 처리하지만, 그 전에 필요한 연결 협상 과정에서 교환되는 정보들을 주고받기 위해 서버가 반드시 필요하다.

Node.js는 이벤트 기반 비동기 처리에 강점을 가지고 있어 실시간으로 여러 클라이언트의 요청을 동시에 처리하는 데 적합하기 때문에 선택되었다. 또한 Express를 통해 기본 서버를 구성하고, 이를 바탕으로 클라이언트의 접속 관리와 요청 라우팅을 안정적으로 수행하였다.

Socket.IO는 이 프로젝트에서 브라우저와 시그널링 서버 간의 실시간 양방향 통신을 담당하였다. 구체적으로는 각 클라이언트가 방에 입장했을 때 이를 알리고, 새로운 피어가 들어오면 Offer를 전달하며, 상대방이 응답하면 Answer를 주고받는 과정이 모두 Socket.IO 이벤트를 통해 이루어졌다. 더불어 ICE 후보 정보도 Socket.IO를 통해 교환되었으며, 이 과정을 통해 브라우저 간 직접 연결이 가능해졌다.

즉, Socket.IO는 WebRTC가 연결을 맺기 위해 필요한 시그널링 데이터를 안정적이고 빠르게 교환하도록 해주는 핵심 역할을 담당하였다.

3.8. 프로젝트 기능 설명

3.8.1. 회원가입/로그인 기능

로그인 화면에서 회원가입 페이지로 이동 시 닉네임, 비밀번호, 비밀번호 확인, 역할 선택 화면을 표시한다.

닉네임의 경우 기존에 저장된 회원의 닉네임과 중복될 경우 에러 메시지를 표시하며 재입력을 요청한다. 또한 입력한 비밀번호와 비밀번호 확인란의 값이 서로 일치하지 않을 경우 역시 에러 메시지를 출력한다.

회원가입을 완료하면 로그인 페이지로 돌아오며 닉네임과 비밀번호를 입력하여 로그인 할 수 있다. 만약 닉네임이 DB에 존재하지 않거나 비밀번호가 틀릴 경우 에러 메시지를 출력하며 재입력을 요청한다

회원가입/로그인 함수

```
/* ---------- 회원가입 ---------- */
const handleSignup = async () => {
  if (!nickname || !password || !confirm) return toast.error("모든 항목을 반드시 기입해주세요");
  if (password !== confirm)               return toast.error("비밀번호가 일치하지 않습니다");

  const userRef = ref(db, `users/${nickname}`);
  const snap    = await get(userRef);
  if (snap.exists())                      return toast.error("이미 존재하는 닉네임입니다");

  const hashed = bcrypt.hashSync(password, bcrypt.genSaltSync(10));
  await set(userRef, { password: hashed, job })
    .then(() => {
      toast.success("회원가입 완료! 로그인 해 주세요");
      setIsLogin(true);
      setPassword(""); setConfirm("");
    })
    .catch(err => alert("저장 오류: "+err.code));
};

/* ---------- 로그인 ---------- */
const handleLogin = async () => {
  if (!nickname || !password) return toast.error("닉네임/비밀번호를 입력해주세요");

  const userRef = ref(db, `users/${nickname}`);
  const snap    = await get(userRef);
  if (!snap.exists())         return toast.error("존재하지 않는 닉네임입니다");

  const user = snap.val();
  if (!bcrypt.compareSync(password, user.password))
    return toast.error("비밀번호가 일치하지 않습니다");
  localStorage.setItem("nickname", nickname); // 로그인 이후 닉네임 저장 처리
  router.push("/rooms");
};
```

3.8.2. 방 생성/참가 기능

화상 통화 프로젝트에서 방은 사용자들이 모여 통신을 시작하는 기본 단위가 된다. 방 생성은 특별히 서버에서 별도의 자원을 준비하는 과정이 아니라, 특정 방 아이디를 기준으로 소켓을 묶는 방식으로 처리된다. 클라이언트가 "join" 이벤트와 함께 방 아이디를 서버로 보내면, 서버는 해당 소켓을 그 방에 참가 시키고, 이미 그 방에 있는 다른 참가자들에게 새로운 사용자가 들어왔음을 알려준다.

방 참가자는 서버를 통해 같은 방에 있는 다른 참가자의 정보를 전달받고, 이후 WebRTC 연결을 위한 Offer, Answer, ICE 후보 교환을 진행한다. 이 과정에서 Socket.IO는 각 참가자가 어떤 방에 속해 있는지를 추적하여 이벤트를 올바른 대상에게 전달한다. 따라서 방 생성과 참가 기능은 결국 Socket.IO의 socket.join(roomId) 기능을 활용하여 자동으로 관리되며, 클라이언트 입장에서는 단순히 특정 방 아이디로 참가 요청을 보내는 것만으로 방이 생성되거나 참여되는 구조다.

방 생성

```
socket.on("create-room", ({ roomId, roomName, password, postContent, nickname }) => {

    rooms[roomId] = {
        roomName,
        password,
        postContent,
        hostId: socket.id,
        hostNickname: nickname,
        users: [{ id: socket.id, nickname }],
        pendingReview: false

    };

    socket.join(roomId);
    broadcastRooms();
});
```

방 참가

```
socket.on("join-room", ({ roomId, password, nickname }) => {
    console.log(`join-room: ${socket.id} -> ${roomId}, pw=${password}`);
    const room = rooms[roomId];
    if (!room || room.pendingReview) return socket.emit("room-not-found");
    if (socket.id !== room.hostId && room.password !== password) {
        console.log(`Invalid password for ${roomId}`);
        return socket.emit("invalid-password");
    }
    if (socket.id === room.hostId) {
        if (!room.users.find(u => u.id === socket.id)) {
            room.users.unshift({ id: socket.id, nickname: room.hostNickname });
        }
    } else {
        room.users.push({ id: socket.id, nickname });
    }
    socket.join(roomId);
    setTimeout(() => {
        console.log(`Emitting room-users for ${roomId}`, JSON.stringify(room.users));
        io.to(roomId).emit("room-users", {
            users: room.users,
            host: room.hostId
        });
    }, 50);
    socket.emit("join-success", { roomId });
    broadcastRooms();
    if (room.users.length === 2) {
        const expert = room.users.find(u => u.id !== room.hostId);
        if (expert) {
            console.log(`Ask host for call permission with ${expert.nickname}`);
            io.to(room.hostId).emit("ask-call-permission", { expertNickname: expert.nickname });
        }
    }
});

```

3.8.3. 통화 연결 허용/거부 기능

방에 사용자가 두 명이 되었을 때, 서버는 방장에게 "새로 들어온 사용자와 통화를 허용할지" 여부를 묻는다. 방장은 allow-call 이벤트를 통해 허용 여부를 서버로 전달한다.

- **허용하는 경우**: 서버는 새로운 사용자(전문가)에게 call-permission-result 이벤트를 보내고, 값은 allow: true로 설정된다. → 전문가 클라이언트는 이후 WebRTC 연결 과정을 진행할 수 있다.

- **거부하는 경우**: 서버는 전문가 클라이언트에게 call-permission-result 이벤트를 allow: false로 보낸 후, 강제로 방에서 내보낸다(force-leave). 이후 방 사용자 목록에서 제거되고, 남아 있는 사용자들에게 갱신된 사용자 목록이 전송된다.

즉, 방장은 최종적으로 들어온 사용자와 화상통화를 시작할지 말지를 결정할 수 있고, 거부 시 상대방은 자동으로 퇴장 처리된다.

```
socket.on("allow-call", ({ roomId, allow }) => {
    const room = rooms[roomId];
    if (!room) return;

    const expert = room.users.find(u => u.id !== room.hostId);
    if (!expert) return;

    if (allow) {
        io.to(expert.id).emit("call-permission-result", { allow: true });
    } else {
        io.to(expert.id).emit("call-permission-result", { allow: false });
        io.to(expert.id).emit("force-leave");

        room.users = room.users.filter(u => u.id !== expert.id);

        io.to(roomId).emit("room-users", {
            users: room.users,
            host: room.hostId
        });
    }
});

```

```jsx
import { io } from "socket.io-client";
import React, { useEffect } from "react";
const socket = io("https://ggg-ar-sol.org");

function CallPermission({ roomId }) {
  useEffect(() => {
    socket.on("call-permission-result", ({ allow }) => {
      if (allow) {
        console.log("통화가 허용되었습니다.");
      } else {
        console.log("통화가 거부되었습니다.");
      }
    });

    socket.on("force-leave", () => {
      console.log("방장이 통화를 거부하여 방에서 퇴장됩니다.");
    });
  }, [roomId]);
  const sendPermission = (allow) => {
    socket.emit("allow-call", { roomId, allow });
  };

  return (
    <div>
      <button onClick={() => sendPermission(true)}>통화 허용</button>
      <button onClick={() => sendPermission(false)}>통화 거부</button>
    </div>
  );
}
export default CallPermission;
```

3.8.4. 화상 통화 기능

화상 통화는 WebRTC(Web Real-Time Communication) 기술을 기반으로 구현된다. WebRTC는 웹 브라우저 간에 별도의 플러그인 없이 음성, 영상 등 미디어를 실시간으로 주고받을 수 있게 하는 강력한 기술이다.

전체 시스템은 크게 시그널링 서버와 클라이언트(Peer) 두 부분으로 구성된다. 시그널링 서버(server/socketServer.js)는 참여자들이 서로를 발견하고 연결 설정을 위한 제어 메시지를 교환하도록 중개하는 역할을 한다. 실제 영상/음성 데이터는 이 서버를 거치지 않는다.

동작 순서 (Step-by-Step)

1. 방 입장 및 사용자 확인 사용자는 특정 방 URL(.../room/방ID)에 접속한다. 클라이언트는 socket.io를 통해 시그널링 서버에 접속하고, join-room 이벤트를 보내 해당 방에 참여 의사를 알린다. 서버는 방에 두 명의 사용자가 참여하면, 두 번째 참여자(전문가)의 입장을 방장에게 알리기 위해 ask-call-permission 이벤트를 보낸다.
2. 통화 시작 (방장 수락) 방장 클라이언트에는 "OO님과 통화를 시작하시겠습니까?" 라는 UI가 표시된다. 방장이 '허용' 버튼을 누르면, 클라이언트는 allow-call 이벤트를 서버로 보낸다. 서버는 이 신호를 받아 전문가 클라이언트에게 call-permission-result (allow: true) 이벤트를 전달하고, 방장 클라이언트는 startHostCall 함수를 호출하여 WebRTC 연결 절차를 시작한다.

3. WebRTC 연결 설정 (시그널링 상세) 이 과정은 WebRTC의 핵심이며, 두 참여자(Peer A: 방장, Peer B: 전문가)가 서로의 정보를 교환하는 과정이다.

3-(A) Peer A (방장) → "Offer" 생성 및 전송

initPeerConnection() 함수를 통해 P2P 연결을 관리하는 주체인 RTCPeerConnection 객체를 생성한다. startHostCall() 함수에서 navigator.mediaDevices.getUserMedia API를 통해 자신의 미디어 스트림을 가져와 RTCPeerConnection 객체에 추가한다 (pc.addTrack()). 이후 createOffer() 메소드를 호출하여 자신의 미디어 정보, 코덱 등이 담긴 Offer(SDP)를 생성하고, setLocalDescription()으로 자신의 연결 정보로 설정한 뒤, 이 Offer를 signal 이벤트에 담아 시그널링 서버로 전송한다.

3-(B) 서버 → "Offer"

중개 서버는 Peer A로부터 받은 signal 이벤트를 같은 방에 있는 Peer B에게 그대로 전달한다.

3-(C) Peer B (전문가) → "Offer" 수신 및 "Answer" 생성/전송

socket.on("signal", ...) 핸들러를 통해 Peer A의 Offer를 수신한다. 수신한 Offer를 setRemoteDescription()으로 상대방의 연결 정보로 설정하고, 자신의 미디어 스트림을 가져와 RTCPeerConnection에 추가한다. 그 다음 createAnswer() 메소드로 응답인 Answer(SDP)를 생성하고 setLocalDescription()으로 자신의 연결 정보로 설정한 후, 이 Answer를 signal 이벤트에 담아 다시 시그널링 서버로 전송한다.

3-(D) 서버 → "Answer"

중개 서버는 Peer B로부터 받은 signal (Answer)을 Peer A에게 그대로 전달한다.

3-(E) Peer A → "Answer" 수신

Peer A는 Peer B의 Answer를 수신하고, setRemoteDescription()으로 상대방(Peer B)의 연결 정보로 최종 설정한다.

4. P2P 연결 수립 (ICE Candidates 교환) SDP 교환이 완료되면, 두 클라이언트는 서로의 미디어 설정을 알게 된다. 이후, 각 클라이언트의 RTCPeerConnection은 onicecandidate 이벤트를 통해 통신 가능한 네트워크 경로(IP 주소, 포트 등)인 ICE Candidate들을 수집하기 시작한다. 수집된 Candidate는 즉시 signal 이벤트를 통해 상대방에게 전송되고, 양측은 addIceCandidate() 메소드로 수신한 후보를 추가한다. 이 과정을 통해 두 클라이언트는 가장 효율적인 통신 경로를 찾아내고 서버를 거치지 않는 직접 P2P 연결을 수립한다.

5. 미디어 스트림 표시 P2P 연결이 성공적으로 수립되면, 각 클라이언트의 RTCPeerConnection에서 ontrack 이벤트가 발생한다. 이 이벤트 핸들러는 상대방으로부터 수신된 미디어 스트림을 <video> HTML 요소의 srcObject에 할당하여 화면에 영상을 표시한다.

6. 통화 종료 사용자가 '통화 종료' 버튼을 누르면 leave-room 이벤트가 서버로 전송된다. RTCPeerConnection 객체가 닫히고(pc.close()) 미디어 스트림이 해제되어 P2P 연결이 종료된다. 서버는 해당 사용자를 방에서 내보내고, 방에 남은 인원이 없으면 방 정보를 삭제한다.

요약

1. Socket.io 서버는 WebRTC 연결에 필요한 제어 정보(SDP, ICE Candidate)를 교환하는 시그널링 채널로만 사용된다.
2. 실제 화상 통화 데이터는 클라이언트 브라우저 간의 P2P 연결을 통해 직접 전송되므로 서버 부하가 적고 지연 시간이 짧다.
3. 클라이언트 코드는 RTCPeerConnection API를 사용하여 미디어 장치 제어, SDP 교환, P2P 연결 수립 등 WebRTC의 복잡한 과정을 처리한다.

3.8.5. AR 주석 추가 기능

AR 주석 기능은 크게 AR 렌더링 및 상호작용을 담당하는 ARComponent.js와 해당 컴포넌트를 페이지에 표시하는 artest/page.js 두 파일의 유기적인 관계로 구현된다. 핵심

로직은 대부분 ARComponent.js에 집중되어 있다.

1. artest/page.js: AR 기능의 컨테이너 역할

이 파일은 AR 기능을 사용자에게 보여주는 페이지의 역할을 한다.

동적 로딩(Dynamic Import)

next/dynamic을 사용하여 ARComponent를 클라이언트 사이드에서만 렌더링(ssr: false)하도록 설정한다. 이는 AR 기능이 브라우저 환경(카메라, WebGL 등)에 크게 의존하기 때문에 서버 사이드 렌더링을 방지하기 위한 필수적인 조치이다.

컴포넌트 배치

페이지 전체를 차지하도록 ARComponent를 렌더링하여 사용자에게 몰입감 있는 AR 경험을 제공한다. page.js의 역할은 비교적 간단하며, 실제 AR 기능의 복잡한 로직은 모두 ARComponent에 위임한다.

2. ARComponent.js: 핵심 AR 로직 및 주석 기능 구현

이 컴포넌트는 MindAR 라이브러리와 A-Frame, Three.js를 사용하여 실제 AR 기능을 구현하는 핵심 부분이다. 주석 추가 기능은 다음 단계로 이루어진다.

<AR 환경 초기 설정>

1. MindAR 및 A-Frame 설정

a-scene 태그를 사용하여 A-Frame 씬(Scene)을 생성한다. mindar-image 속성을 통해 이미지 타겟(targets.mind)을 지정하고, autoStart: false 로 설정하여 수동으로 AR 엔진을 시작할 수 있도록 한다. raycaster 속성은 .target-plane 클래스를 가진 객체에만 광선(ray)이 반응하도록 설정하여, 이미지 타겟 위를 클릭했을 때만 상호작용이 일어나도록 제한한다.

2. 카메라 스트림 및 캔버스 결합

useEffect 훅 안에서 navigator.mediaDevices.getUserMedia를 호출하여 카메라 영상 스트림을 가져온다. 가져온 영상은 보이지 않는 <video> 태그에 출력하고, 이 영상과 A-Frame이 렌더링하는 AR 오버레이(a-scene의 캔버스)를 combinedCanvasRef라는 별도의 캔버스에 매 프레임마다 함께 그린다. 이 방식을 통해 최종적으로 사용자에게는 실제 카메라 영상 위에 3D 객체가 증강된 것처럼 보이는 화면을 제공하며, 이 결합된 영상 스트림은 WebRTC를 통해 다른 참여자에게 전송될 수 있다.

<주석(Annotation) 추가 로직>

주석 추가는 사용자가 이미지 타겟 위를 클릭했을 때 발생하며, handleClick 함수가 이

로직을 담당한다.

1. 클릭 이벤트 감지

a-plane으로 생성된 .target-plane 객체에 click 이벤트 리스너를 추가한다. 이 평면은 이미지 타겟과 동일한 크기로 보이지 않게(opacity: 0.1) 겹쳐져 있어, 사용자가 타겟을 클릭하는 것을 감지하는 역할을 한다.

2. 3D 좌표 계산

클릭 이벤트가 발생하면, 이벤트 객체(event.detail.intersection.point)로부터 월드(World) 3D 좌표를 얻을 수 있다. parent.object3D.worldToLocal() 함수를 사용하여 이 월드 좌표를 이미지 타겟의 로컬(Local) 좌표로 변환한다. 이렇게 해야 주석이 타겟을 기준으로 정확한 위치에 생성된다.

3. 선택된 도구에 따른 객체 생성

handleClick 함수는 상위 컴포넌트에서 props로 전달받은 selectedTool의 값에 따라 다른 종류의 주석을 생성한다. switch (currentTool) 문을 통해 분기 처리를 한다.

- marker: a-cone 엘리먼트를 생성하여 파란색 역삼각뿔 형태의 마커를 추가한다.
- text: prompt 창으로 사용자에게 텍스트를 입력받은 후, a-entity 컨테이너 안에 a-text 엘리먼트를 두 개(하나는 테두리 효과용) 생성하여 3D 텍스트를 추가한다.
- cpu, ram, gpu: a-entity에 gltf-model 속성을 부여하여 미리 준비된 3D 모델(public/models/폴더)을 불러와 씬에 추가한다.

4. 객체를 씬에 추가

생성된 A-Frame 엘리먼트(주석)를 클릭된 타겟(parent)의 자식으로 추가(parent.appendChild(cone))한다. 이로써 주석은 이미지 타겟에 종속되어 타겟이 움직이면 함께 따라 움직이게 된다.

<다른 참여자(Peer)의 주석 처리>

WebRTC를 통해 다른 참여자로부터 주석 생성 이벤트(peerClickCoords)를 받으면, 로컬 환경에서도 동일한 주석을 생성한다.

1. 좌표 변환

Peer가 클릭한 2D 화면 좌표(peerClickCoords)를 get3DPoint 함수를 통해 3D 공간 좌표로 변환한다. 이 함수는 THREE.Raycaster를 사용하여 카메라 시점에서 클릭 지점으로 광선을 쏘아 이미지 타겟 평면과의 교차점을 찾는다.

2. 객체 생성

handleClick과 유사하게 peerTool의 종류에 따라 marker, text, gltf-model 등 해당하는 3D 객체를 생성하여 씬에 추가한다. Peer가 생성한 객체는 식별을 위해 다른 색상(예: marker는 빨간색)으로 표시된다.

요약

1. artest/page.js가 ARComponent를 렌더링하여 AR 환경을 초기화한다.
2. ARComponent는 MindAR를 통해 이미지 타겟을 인식하고 카메라 영상 위에 A-Frame 씬을 오버레이한다.
3. 사용자가 상위 UI에서 특정 도구(마커, 텍스트 등)를 선택한다.
4. 사용자가 AR로 보이는 이미지 타겟 위를 클릭하면, handleClick 함수가 호출된다.
5. handleClick 함수는 클릭된 3D 위치를 계산하고, 선택된 도구에 맞는 3D 객체(주석)를 생성하여 이미지 타겟에 종속시킨다.
6. 생성된 주석 정보는 소켓을 통해 다른 참여자에게 전송되고, 다른 참여자들의 화면에도 동일한 위치에 주석이 생성되어 실시간 협업이 이루어진다.

이러한 구조를 통해 사용자는 직관적으로 AR 공간에 정보를 추가하고 다른 사람과 공유할 수 있게 된다.

3.8.6. 전문가 상담 리뷰/열람 기능

전문가와의 화상 통화가 종료된 후, 사용자는 자동으로 전문가에 대한 리뷰를 작성할 수 있는 별도의 페이지로 이동할 수 있다. 이 리뷰 작성 화면에서는 전문가의 상담에 대해 자신의 소감을 자유롭게 입력하는 텍스트 후기와 함께, 서비스의 만족도 또는 상담의 전문성 등에 대해 1점부터 5점까지의 별점도 남길 수 있도록 구성되어 있다.

리뷰를 모두 작성한 뒤 제출하면, 해당 정보는 Firebase DB에 저장된다. 이를 통해 모든 사용자는 자신이 작성한 과거의 리뷰 내역을 별도의 리뷰 확인 페이지에서 다시 조회할 수 있고, 전문가 역시 본인과 통화한 사용자들이 남긴 다양한 리뷰 내역을 한눈에 열람할 수 있어, 자신에 대한 객관적인 평가와 피드백을 쉽게 확인하는 것이 가

능하다.

리뷰 확인 시스템 함수

```
export default function ShowReview() {
    const db = getDatabase(app);
    const router  = useRouter();
    const [reviewUser, setReviewUser] = useState(null);
    const [reviews, setReviews] = useState([]);
    const [userJob, setUserJob] = useState(null);

    useEffect(() => {
        setReviewUser(localStorage.getItem("nickname"));
    }, [])

    useEffect(()=> {
        const fetchJob = async () => {
        const snapshot = await get(ref(db, `users/${reviewUser}/job`));
        setUserJob(snapshot.val());
        }
        fetchJob();
    }, [reviewUser]);

    useEffect(() => {
        if (!reviewUser) return;
        const reviewsRef = ref(db, "reviews");
        get(reviewsRef).then((snap) => {
        if (!snap.exists()) {
            setReviews([]);
            return;
        }

        const filtered = Object.entries(snap.val())
            .filter(([_, v]) => v.expert === reviewUser || v.user === reviewUser)
            .map(([id, v]) => ({ id, ...v }));

        setReviews(filtered);
        });
    }, [db, reviewUser]);
```

리뷰 작성 시스템 함수

```
export default function Review() {
    const db = getDatabase(app);
    const router  = useRouter();
    const [user, setUser] = useState(null);
    const [expert, setExpert] = useState(null);
    const [reviewSerialNum, setReviewSerialNum] = useState(null);
    const [reviewContent, setReviewContent] = useState("");
    const [star, setStar] = useState(0);

    useEffect(() => {
        setUser(localStorage.getItem("nickname"));
        setExpert(localStorage.getItem("expert"));
        setReviewSerialNum(localStorage.getItem("id"));
    }, [])

    useEffect(() => {
        return () => {
            window.localStorage.removeItem("id");
            window.localStorage.removeItem("expert");
        };
    }, []);

    const initialize = () => {
        setStar(0);
        setReviewContent("");
        setExpert(null);
        setReviewSerialNum(null);
        setUser(null);
        window.localStorage.removeItem("id");
        window.localStorage.removeItem("expert");
        router.push('/rooms');
    };

    const storeReview = async () =>{
        const reviewRef = ref(db, `reviews/${reviewSerialNum}`);
        await set(reviewRef, {user, expert, reviewContent, rating: star})
        .then(() => {
            toast.info("리뷰 작성이 완료되었습니다");
            initialize();
        })
        .catch(err => alert("저장 오류: "+err.code));
    };
```

3.8.7. 화면 녹화 및 열람 기능

사용자와 전문가가 모두 하나의 세션에 접속하면, 양쪽 클라이언트 모두에게 화면 녹화 기능을 사용할지 여부를 선택할 수 있는 옵션이 제공된다. 각 사용자는 녹화를 시작하기 전에 화면 녹화 기능의 활성화 여부를 직접 결정할 수 있으며, 만약 녹화를 시작하면 영상통화 또는 AR 협업 세션의 모든 화면이 실시간으로 기록된다.

녹화가 시작된 이후에는, 통화가 진행되는 동안의 모든 시각적 내용과 오디오가 함께 캡처된다. 통화가 종료되면, 그동안 녹화된 영상 데이터는 웹 브라우저의 MediaRecorder API를 통해 webm 형식으로 인코딩된다. 파일은 'YYYYMMDDHHMM.webm' 형식의 고유한 이름으로 Supabase DB에 저장된다. 이를 통해 영상 파일의 생성 시각과 세션이 한눈에 구분 가능하다.

세션에 참여했던 각 클라이언트는 별도의 녹화 영상 확인 페이지에 접속할 수 있다. 해당 페이지에서는 데이터베이스에 저장된 자신의 녹화 영상 목록을 확인할 수 있으

며, 원하는 영상을 선택해 재생할 수 있다. 이러한 방식으로, 사용자는 자신이 직접 녹화한 협업 세션 결과물을 필요할 때마다 쉽게 열람하거나 복습할 수 있다. 즉, 녹화 파일을 통해 원격 협업, 피드백 검토, 전문 상담 내용 기록 등 다양한 목적에 맞게 영상을 적극적으로 활용할 수 있도록 지원된다.

화면 녹화 함수

```
async function startRecording() {

  const ok = window.confirm("화면 녹화를 시작할까요?");
  if (!ok) return;

  if (recorderRef.current) return;

  const displayStream = await navigator.mediaDevices.getDisplayMedia({
    video: {
      displaySurface: "window",
      preferCurrentTab: true,
      surfaceSwitching: "exclude",
      selfBrowserSurface: "include"
    },
    audio: true
  });

  const recorder = new MediaRecorder(displayStream, {
    mimeType: "video/webm;codecs=vp9,opus",
  });

  recorder.ondataavailable = (e) => {
    if (e.data.size) recordedChunks.current.push(e.data);
  };

  recorder.onstop = async () => {
    const blob = new Blob(recordedChunks.current, { type: 'video/webm' });
    recordedChunks.current = [];
    const ts = new Date().toISOString().slice(0, 16).replace(/[-:]/g, '');
    const currentU = localStorage.getItem("nickname");
    const fileName = `${ts}_${id}.webm`;
    const filePath = `${currentU}/${fileName}`;

    const { error } = await supabase.storage
      .from('recordings')
      .upload(filePath, blob, {
        contentType: 'video/webm',
        upsert: false
      });

    if (error) {
      toast.error(`Supabase 업로드 실패: ${error.message}`);
      return;
    }


    const { data } = supabase.storage
      .from('recordings')
      .getPublicUrl(filePath);

    const videoUrl = data.publicUrl;
    toast.success('Supabase 업로드 완료!');


  };

  recorder.start();
  recorderRef.current = recorder;
};

function stopRecording() {
  if (recorderRef.current) {
    recorderRef.current.stop();
    recorderRef.current = null;
  }
};
```

녹화 영상 확인 함수

```
export default function ShowRecording() {
    const [videos, setVideos] = useState([]);
    const router = useRouter();

    function formatDate(name) {
        try {
            const raw = name.split("_")[0];
            const y = raw.slice(0, 4);
            const m = raw.slice(4, 6);
            const d = raw.slice(6, 8);
            const h = raw.slice(9,11);
            const t = raw.slice(11,13);
            return `${y}년 ${m}월 ${d}일 ${h}시 ${t}분`;
        } catch {
            return name;                                    // 예상 형식이 아니면 그대로
        }
    }

    useEffect(() => {
        const nickname = localStorage.getItem("nickname");
        if (!nickname) return;

        async function fetchVideos() {

        const { data, error } = await supabase.storage
            .from("recordings")
            .list(nickname, {
            limit: 100,
            sortBy: { column: "name", order: "desc" },
            });

        if (error) {
            console.error("영상 목록 불러오기 실패:", error.message);
            return;
        }

        // 각 파일에 대한 public URL 생성
        const items = await Promise.all(
            data.map(async (file) => {
            const { data: urlData } = supabase.storage
                .from("recordings")
                .getPublicUrl(`${nickname}/${file.name}`);
            return { name: file.name, url: urlData.publicUrl };
            })
        );

        setVideos(items);
        }

        fetchVideos();
    }, []);

    const left = () => {
        router.push('/rooms')
    };
```

4. 연구 결과 분석 및 평가

4.1. ARSOL 프로젝트 보완 및 대응 방안 계획

서비스 시나리오 리팩토링: '전문가 주도' 상담 모델로 전환

목표: 현재의 '고객이 방 생성 후 대기'하는 비효율적인 모델을 '전문가가 방을 생성하고 고객을 초대'하는 직관적인 모델로 변경하여 서비스 품질을 향상시킨다.

실행 계획:

app/rooms/page.js (상담 방 게시판 페이지)

- 수정: '새 상담 방 만들기' 버튼이 '전문가' 역할의 사용자에게만 보이도록 로직을 변경한다.
- 수정: 기존의 방 목록 전체가 '전문가'에게만 보이도록 변경한다. '사용자' 역할

의 계정에게는 더 이상 불필요한 방 목록이 노출되지 않도록 한다.

app/create/page.js (방 만들기 페이지)

- 수정: 전문가가 방을 만드는 시점에는 고객의 '게시물 내용'이 불필요하므로 해당 입력 필드(textarea)와 관련 상태(state), 생성 로직을 모두 제거하여 페이지를 간소화한다.

server/socketServer.js (실시간 서버)

- 수정: create-room 이벤트 핸들러에서 더 이상 postContent 파라미터를 받지 않도록 수정한다.
- 수정: getRoomList 함수에서 클라이언트로 방 목록을 보낼 때, postContent 데이터를 제외하도록 수정한다.

신규 기능 (고객용 접속)

- 추가: 고객이 전문가로부터 받은 고유 접속 링크(URL)를 입력하여 바로 상담 방에 참여할 수 있는 간단한 페이지 또는 컴포넌트를 app/page.js나 별도의 페이지에 추가한다.

UI/UX 개선

목표: 자문 내용에 따라 사용자의 편의성을 증대시킨다.

실행 계획:

app/room/[id]/page.js (원격 상담 페이지)

- 수정: CSS 스타일을 조정하여, 전문가의 화면보다 가이드를 받는 '고객' 자신의 카메라 화면이 더 크게 보이도록 레이아웃을 변경한다.

AR 기능 현실화 및 안정성 확보

목표: 기술적 난이도가 매우 높은 3D 모델 AR 기능의 부담을 줄이고, 더 안정적이고 실용적인 대안을 적용한다.

실행 계획:

app/components/ARComponent.js (AR 기능 컴포넌트)

- 분석 및 수정: 현재 3D 모델 기반 AR 기능의 완성도와 안정성을 평가한다.
- 대안 제시: 자문 내용에 따라 복잡한 3D 모델 추적 대신 안정적으로 구현 가능한 'AR 드로잉'이나 'AR 화살표 포인터' 기능으로 단순화하여 사용성과 안정성을 높이는 방향으로 코드를 수정한다.

서비스 확장성 문서화

목표: 향후 기능 확장을 대비하여 현재 아키텍처의 명확한 한계와 확장 방안을 문서로 명시한다.

실행 계획:

README.md (프로젝트 문서)

- 추가: 현재 시스템은 WebRTC 기반의 P2P(Peer-to-Peer) 연결을 사용하며, 이는 '1:1 상담'에 최적화되어 있음을 명시한다.
- 추가: 만약 추후 '다자간 협업'으로 서비스를 확장할 경우, 별도의 미디어 서버(SFU/MCU) 구축이 필요하다는 아키텍처 가이드를 추가한다.

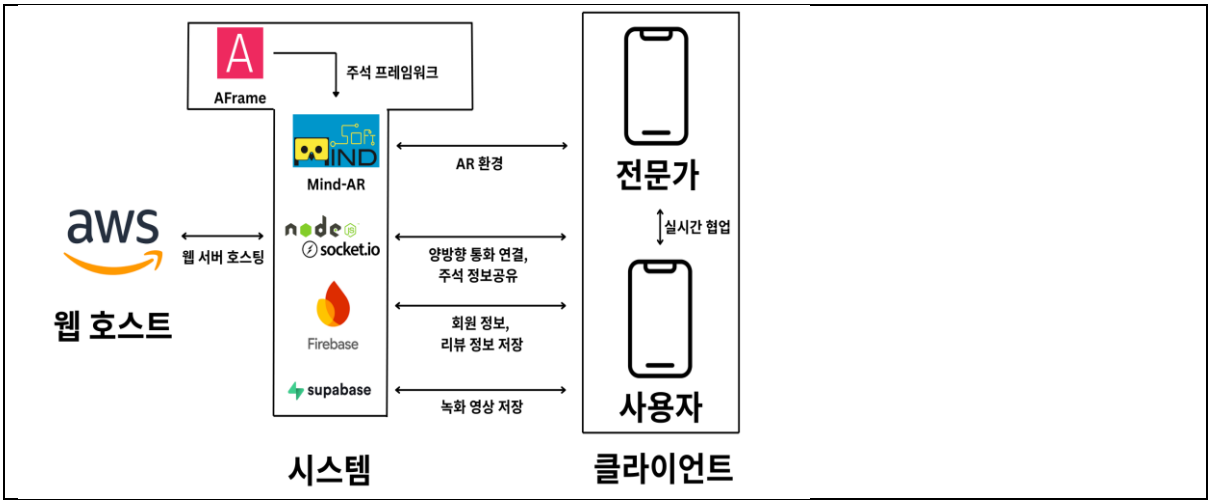
4.2. 개발 일정

개발 구분	세부 항목	5월	6월	7월	8월	9월
기획	사전 조사					
	Pre-Production 회의					
	Production 회의					
AR 기능 구현	AR 환경 구축					
	AR 주석 생성 기능 구현					
	AR 주석 클라이언트 간 동기화					
DB 설계	회원가입, 로그인					
	전문가 리뷰					
	상담 내역 기록					
통화 연결 시스템	게시판 서버 구축					
	WebRTC 서버 구축					
UI/UX 디자인	전반적 Layout 디자인					

4.3. 역할 분담

이름	맡은 일
강유승	DB 설계 통화 화면 녹화 리뷰 시스템 주석 3D 모델링 AR 주석 선택 및 동기화
김준성	AR 전환 기능 타겟 이미지 인식 및 피드백 AR 주석 원격 생성 및 이동
윤민혁	UI 디자인 상담 게시판 서버 구축 화상통화 환경 구축 도메인 생성, 서비스 배포 및 관리

4.4. 시스템 구성도



4.5. API 명세서

본 명세서는 (클라이언스 -> 서버 / 서버 -> 클라이언트 / http api) 세 종류를 다룬다.

클라이언트 -> 서버 (Client -> Server)

이벤트: create-room
설명: 사용자가 새로운 화상 통화 방을 생성한다.
페이로드: { roomId, roomName, password, postContent, nickname }

- roomId: (String) 고유한 방 ID
- roomName: (String) 방 제목
- password: (String) 방 비밀번호 (선택 사항)
- postContent: (String) 방 관련 게시물 내용
- nickname: (String) 생성자(방장)의 닉네임

이벤트: get-rooms
설명: 현재 활성화된 모든 방의 목록을 서버에 요청한다.
페이로드: 없음

이벤트: join-room
설명: 사용자가 기존에 생성된 방에 참여한다.
페이로드: { roomId, password, nickname }

- roomId: (String) 참여할 방의 ID
- password: (String) 방의 비밀번호
- nickname: (String) 참여자의 닉네임

이벤트: allow-call
설명: 방장이 전문가의 화상 통화 요청을 수락하거나 거부한다.
페이로드: { roomId, allow }

- roomId: (String) 현재 방의 ID
- allow: (Boolean) 통화 허용 여부 (true/false)

이벤트: signal
설명: WebRTC 연결 설정을 위한 시그널링 데이터를 서버를 통해 다른 클라이언트에게 전달한다. (SDP Offer/Answer, ICE Candidate 등)

페이로드: { roomId, data }

- roomId: (String) 현재 방의 ID
- data: (Object) WebRTC 시그널링 데이터

이벤트: ar-mode-change
설명: 클라이언트가 AR 모드로 전환했음을 서버에 알린다.
페이로드: { roomId, arMode }

- roomId: (String) 현재 방의 ID
- arMode: (Boolean) AR 모드 활성화 여부

이벤트: peer-click
설명: (전문가) 상대방의 영상 화면을 클릭했을 때, 해당 좌표를 방장에게 전송하여 AR 공간에 객체를 배치하도록 요청한다.
페이로드: { roomId, coords }

- roomId: (String) 현재 방의 ID
- coords: (Object) 클릭된 좌표 { x, y }

이벤트: peer-select
설명: (전문가) 주석 도구를 선택했음을 방장에게 알린다.
페이로드: { roomId, tool, text }

- roomId: (String) 현재 방의 ID
- tool: (String) 선택한 도구의 종류 ('marker', 'text' 등)
- text: (String) 텍스트 도구 사용 시 입력한 내용

이벤트: leave-room
설명: 사용자가 방을 나간다.
페이로드: (String) 나가는 방의 ID

이벤트: delete-room
설명: 방장이 방을 삭제한다.
페이로드: (String) 삭제할 방의 ID

AR 주석 및 객체 조작 (Annotation & Object Manipulation)
이벤트: annotation-added
설명: (방장) AR 공간에 새로운 주석(마커, 텍스트, 3D 모델 등)이 추가되었음을 서버로

전송하여 전문가에게 전달하도록 한다.
페이로드: { roomId, annotation }

- roomId: (String) 현재 방의 ID
- annotation: (Object) 추가된 주석 정보 { id, type }

이벤트: delete-annotation
설명: (전문가) 특정 주석을 삭제할 것을 방장에게 요청한다.
페이로드: { roomId, annotationId }

- roomId: (String) 현재 방의 ID
- annotationId: (String) 삭제할 주석의 ID

이벤트: delete-all-annotations
설명: (전문가) 모든 주석을 삭제할 것을 방장에게 요청한다.
페이로드: { roomId }

이벤트: request-object-transform
설명: (전문가) 특정 주석 객체의 현재 위치/회전 값을 방장에게 요청한다. (객체 이동 시작 시)
페이로드: { roomId, objectId }

- objectId: (String) 정보를 요청할 객체의 ID

이벤트: update-object-transform
설명: (전문가) 특정 주석 객체의 위치/회전 값을 변경하도록 방장에게 요청한다. (객체 이동 완료 시)
페이로드: { roomId, objectId, position, rotation }

- objectId: (String) 변경할 객체의 ID
- position: (Object) 새로운 위치 값 { x, y, z }
- rotation: (Object) 새로운 회전 값 { x, y, z }

이벤트: send-object-transform
설명: (방장) request-object-transform 요청에 대한 응답으로, 특정 주석 객체의 현재 위치/회전 값을 서버로 전송한다.
페이로드: { roomId, objectId, position, rotation }

- objectId: (String) 정보 요청을 받은 객체의 ID
- position: (Object) 현재 위치 값 { x, y, z }

- rotation: (Object) 현재 회전 값 { x, y, z }

서버 -> 클라이언트 (Server -> Client)

이벤트: rooms-updated
방향: 서버 -> 모든 클라이언트
설명: 방 목록에 변경사항(생성, 삭제, 인원 변동)이 생겼을 때, 새로운 방 목록을 모든 클라이언트에게 전송한다.
페이로드: (Array) 방 객체 목록 [{ id, roomName, postContent, count }]

이벤트: room-not-found
방향: 서버 -> 특정 클라이언트
설명: 참여하려는 방이 존재하지 않을 경우 전송된다.
페이로드: 없음

이벤트: invalid-password
방향: 서버 -> 특정 클라이언트
설명: 방 입장 시 비밀번호가 틀렸을 경우 전송된다.
페이로드: 없음

이벤트: room-users
방향: 서버 -> 해당 방의 모든 클라이언트
설명: 방의 사용자 목록에 변경이 있을 때마다 업데이트된 목록을 전송한다.
페이로드: { users, host }

- users: (Array) 현재 방에 있는 사용자 객체 목록 [{ id, nickname }]
- host: (String) 방장의 소켓 ID

이벤트: join-success
방향: 서버 -> 특정 클라이언트
설명: 방에 성공적으로 입장했음을 알린다.
페이로드: { roomId }

이벤트: ask-call-permission
방향: 서버 -> 방장 클라이언트
설명: 전문가가 방에 입장했을 때, 방장에게 통화 시작 여부를 묻기 위해 전송된다.
페이로드: { expertNickname }

- expertNickname: (String) 입장한 전문가의 닉네임

이벤트: call-permission-result
방향: 서버 -> 전문가 클라이언트
설명: 방장이 통화 요청을 수락 또는 거부했음을 전문가에게 알린다.
페이로드: { allow }

- allow: (Boolean) 통화 허용 여부

이벤트: force-leave
방향: 서버 -> 전문가 클라이언트
설명: 방장이 통화를 거부했을 때, 전문가를 방에서 내보내기 위해 전송된다.
페이로드: 없음

이벤트: signal
방향: 서버 -> 해당 방의 다른 클라이언트
설명: 한 클라이언트로부터 받은 WebRTC 시그널링 데이터를 상대방 클라이언트에게 중계한다.
페이로드: { from, data }

- from: (String) 데이터를 보낸 클라이언트의 소켓 ID
- data: (Object) WebRTC 시그널링 데이터

이벤트: peer-ar-mode-changed
방향: 서버 -> 해당 방의 다른 클라이언트
설명: 상대방의 AR 모드가 변경되었음을 알린다.
페이로드: { arMode }

- arMode: (Boolean) 상대방의 AR 모드 활성화 여부

이벤트: place-object
방향: 서버 -> 방장 클라이언트
설명: 전문가가 클릭한 좌표를 전달하여, 방장의 AR 화면에 객체를 배치하도록 한다.
페이로드: { coords }

이벤트: tool-select
방향: 서버 -> 방장 클라이언트
설명: 전문가가 선택한 주석 도구를 방장에게 전달한다.

페이로드: { tool, text }

이벤트: room-closed
방향: 서버 -> 해당 방의 모든 클라이언트
설명: 방장이 방을 닫았음을 알린다.
페이로드: 없음

이벤트: peer-disconnected
방향: 서버 -> 해당 방의 클라이언트
설명: 상대방의 연결이 끊어졌음을 알린다.
페이로드: 없음

AR 주석 및 객체 조작 (Annotation & Object Manipulation)
이벤트: annotation-added
방향: 서버 -> 전문가 클라이언트
설명: 방장의 AR 공간에 새로운 주석이 추가되었음을 전문가에게 알린다.
페이로드: (Object) 추가된 주석 정보 { id, type }

이벤트: delete-annotation
방향: 서버 -> 방장 클라이언트
설명: 전문가가 요청한 주석 삭제 명령을 방장에게 전달한다.
페이로드: { annotationId }

이벤트: delete-all-annotations
방향: 서버 -> 방장 클라이언트
설명: 전문가가 요청한 모든 주석 삭제 명령을 방장에게 전달한다.
페이로드: 없음

이벤트: request-object-transform
방향: 서버 -> 방장 클라이언트
설명: 전문가가 요청한 특정 주석 객체의 현재 위치/회전 값 요청을 방장에게 전달한다.
페이로드: { objectId }

이벤트: update-object-transform

방향: 서버 -> 방장 클라이언트
설명: 전문가가 변경한 특정 주석 객체의 새로운 위치/회전 값을 방장에게 전달한다.
페이로드: { objectId, position, rotation }

이벤트: send-object-transform
방향: 서버 -> 전문가 클라이언트
설명: 방장으로부터 받은 특정 주석 객체의 현재 위치/회전 값을 전문가에게 전달한다.
페이로드: { objectId, position, rotation }

HTTP API (Next.js API Routes)

이 API들은 Next.js의 API 라우트 기능을 통해 제공되는 표준 HTTP 엔드포인트이다. 주로 실시간 통신이 필요 없는 초기 데이터 로딩이나 방 생성 요청에 사용된다.

엔드포인트: /api/rooms
메서드: GET
설명: 현재 생성된 모든 방의 기본 정보 목록을 조회한다.
요청: 없음
응답 (성공):

- Status: 200 OK
- Body: (Array) 방 객체 목록 [{ id, count }]
- id: (String) 방의 고유 ID
- count: (Number) 해당 방의 참여자 수

엔드포인트: /api/createRoom
메서드: POST
설명: 새로운 방을 생성(등록)한다. 클라이언트는 이 API로 방을 먼저 등록한 후, 해당 방 ID를 가지고 소켓 연결 및 방 참여를 시작할 수 있다.
요청:

- Header: Content-Type: application/json
- Body: { "roomId": "..." }
- roomId: (String) 생성할 방의 고유 ID

응답 (성공):

- Status: 200 OK
- Body: { "success": true, "roomId": "..." }

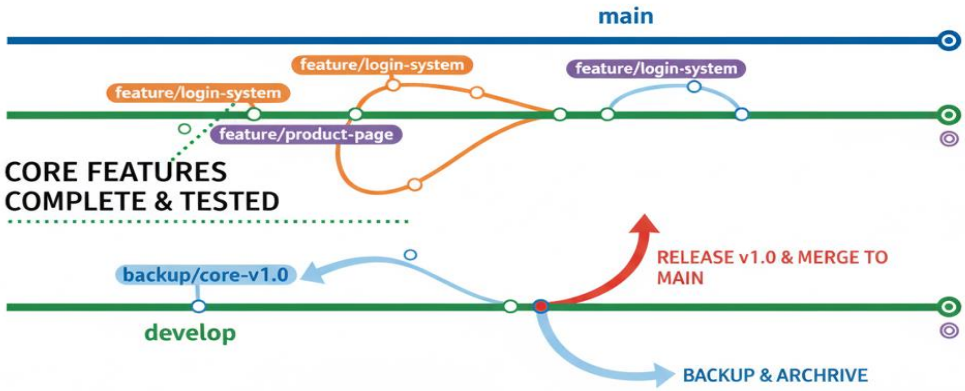
응답 (실패):

- Status: 405 Method Not Allowed (POST가 아닌 다른 메서드로 요청 시)

4.6. 협업 방식

협업 방식

각 피처를 구현할 때 Branch를 따로 생성하여 작업 후 기능 테스트가 완료되면 Develop에 Merge, 핵심 기능들이 완성될 때마다 BackUp Branch 생성 후 main에 Merge하는 방식으로 작업하였다



5. 결론 및 향후 연구 방향

5.1. 결론 및 기대효과

본 프로젝트를 통해 웹 브라우저 기반 AR 원격 협업 시스템을 개발하는데 성공하였다. 사용자는 별도의 애플리케이션 설치 없이 웹 브라우저만으로 바로 AR 협업 기능에 접근 가능하며, WebRTC를 통한 화상통화와 실시간 데이터 공유가 원활하게 작동한다. 특히, 마커 기반 이미지 인식과 3D 주석 기능을 결합하여 직관적이고 몰입감 높은 원격 지원이 가능하다.

이 시스템은 컴퓨터 조립과 같은 전문 지식 없이도 사용자가 실시간 맞춤형 지원을

받을 수 있게 하여 조립 과정의 어려움과 시간 소요를 크게 감소시키는 효과가 기대된다. 또한 AWS를 통한 안정적인 배포와 Socket.io 기반의 실시간 통신으로 다중 사용자 간 협업 환경 구축이 용이하다.

5.2. 향후 연구 방향

향후 연구는 기술 고도화 및 사용자 경험 개선에 중점을 둔다.

기존 주석 모델 개선 및 3D 오브젝트 다양화: 향후 연구에서는 다양한 형태와 기능을 가진 3D 모델을 추가하여 사용자가 더욱 풍부하고 직관적인 정보를 증강현실 환경에서 표현할 수 있도록 개선한다.

다양한 플랫폼 호환성 확대: 모바일 및 다양한 운영체제 환경에 맞춘 최적화 작업과 네이티브 앱과의 연동 기술을 개발한다.

보안성 강화 및 개인정보 보호: 데이터 암호화, 인증 강화 등 사용자 데이터 보호를 위한 보안 기술 접목을 우선 과제로 진행한다.

사용자 인터페이스 및 사용성 연구: 사용자의 피드백을 반영하여 UI/UX를 지속적으로 개선하고, AR 경험을 보다 직관적이고 접근하기 쉽게 만든다.

6. 참고 문헌

논문 내용에 직접 관련이 있는 문헌에 대해서는 관련이 있는 본문 중에 참고문헌 번호를 쓰고 그 문헌을 참고문헌란에 인용 순서대로 기술한다. 참고문헌은 영문으로만 표기하며 학술지의 경우에는 저자, 제목, 학술지명, 권, 호, 쪽수, 발행년도의 순으로, 단행본은 저자, 도서명, 발행소, 발행년도의 순으로 기술한다.