Innovation Academy

「2025 IA x AI 해커톤」

개발 완료 보고서

팀　　명 : Samadhi

프로젝트명 : 실시간 요가 자세 분석 서비스

※ 유의사항

1. 본 보고서의 내용은 최대 2 page이내로 작성
2. 보고서의 설명을 보충하기 위해 필요한 **사진 또는 그래프 첨부** 가능
3. 제출 서류는 일체 반환을 하지 않음
4. 제출 파일명 작성 요령
   - 파일명: [2025 IA x AI 해커톤]_팀명
5. 서체: 맑은고딕, 크기: 12p, 줄간격: 160%
6. 제출처: 깃허브에 업로드

Innovation Academy

「2025 IA x AI 해커톤」

프로젝트명	실시간 요가 자세 분석 서비스 : Samadhi
프로젝트 목표	사용자가 자유롭게 요가 영상을 선택하고, 웹캠을 통해 촬영한 자신의 요가 자세를 실시간으로 분석하여 영상 속 강사의 자세와 비교하는 서비스 개발을 목표로 한다. MediaPipe를 활용해 사용자 관절 좌표를 추출하고, 코사인 유사도 기반 알고리즘을 통해 동작 유사도를 계산한다. 이를 바탕으로 사용자는 자신의 자세와 강사의 자세 간 차이를 직관적으로 확인할 수 있으며, 운동 기록에 대한 점수화 및 관리 기능을 통해 사용자가 운동 성과를 추적할 수 있도록 한다. 구체적인 세부 목표는 다음과 같다. 1. 휴먼 포즈 추정: MediaPipe를 활용한 33개 랜드마크 실시간 검출 2. 자세 분류: 주요 요가 동작 20개 자동 인식 및 분류 시스템 구축 3. 동작 유사도 분석: 코사인 유사도 기반 정량적 유사도 계산 기능 4. 서비스 구현: 웹캠 입력, 영상 처리, 운동 기록 관리 기능을 갖춘 웹 어플리케이션 개발
개발 환경	- FE: Next.js 15 (App Router), TypeScript - AI/ML: MediaPipe Tasks Vision - BE: Spring Boot 3.3.2 - DB: MySQL - Cloud: AWS (ECR, EC2, S3)
구현 기능	1. 휴먼 포즈 추정 먼저, MediaPipe 포즈 랜드마크를 활용하여 사용자의 신체에서 33개의 관절 포인트를 실시간으로 검출하며, 각 관절의 3D 좌표(x, y, z)와 가시성(visibility) 정보로 정확한 자세 분석의 기반을 마련했다. 또한 웹캠 입력 및 비디오 파일 처리 파이프라인을 구현하고, 관절 데이터 전처리 및 정규화를 통해 실시간 렌더링 및 시각화를 구성했다. 2. 자세 분류 벡터화된 자세 데이터베이스를 구축하여 40가지 요가 자세(Cat, Tree, Downward Dog, Warrior 시리즈 등)를 자동으로 분류한다. 각 자세는 사전 학습된 관절 각도 벡터와의 비교를 통해 90점 이상의 유사도를 가질 때 인식된다. 3. 동작 유사도 분석 사용자의 자세와 참고 자세를 비교하기 위한 코사인 유사도 측정 방식을

	구현했다. 자세의 방향성과 각도 관계를 비교하여 자세의 형태적 유사성을 평가하며 최종 점수는 0-100% 범위로 정규화되고, 점수 범위에 따라 적절한 피드백 문구를 사용자에게 제공할 수 있도록 구성했다. 4. 서비스 구현 우선 홈화면과 로그인, 회원가입 기능을 구현했다. 홈화면에서 '운동 시작하기' 버튼을 누르면 운동 준비 프로세스가 시작된다. 운동 준비 프로세스는 4단계로 이루어진다. 먼저 샘플 영상 모드 또는 화면 공유 모드 중 하나를 선택한다. 샘플 영상 모드는 시스템에서 제공하는 전문가의 요가 동작 영상을 활용한 운동 모드이고, 화면 공유 모드는 사용자가 선호하는 영상을 틀고 화면 공유를 활용해서 운동이 가능한 모드이다. 이를 통해 다양한 요가 콘텐츠 활용이 가능하여 사용자의 선택권을 확대하고자 했다. 그 후 웹캠 설정이 이루어진다. 브라우저에서 웹캠 접근 권한을 허용하면, 웹캠 미리보기가 뜨게 된다. 최종적으로 선택한 영상과 웹캠 설정을 최종 확인하고 준비가 완료되면 운동 페이지로 이동하게 된다. 운동 진행 중 화면의 경우, 화면을 좌우로 분할하여 왼쪽에는 참고 영상, 오른쪽에는 사용자의 웹캠 화면을 표시한다. 실시간 유사도 점수를 기반으로 한 피드백 문구를 우측 하단에 표시하여 즉각적인 피드백을 제공하고, 왼쪽 하단에는 참고 영상에서 분류된 자세를 바탕으로 자세 타임라인별 유사도가 표시된다. 운동 중 인식된 자세가 변경될 때마다 자동으로 타임라인 구간을 생성하고 각 구간은 자세 이름, 시작/종료 시간, 해당 자세에 대한 평균 유사도 점수를 포함한다. 또한 사용자 편의성을 위해 설정에서 각 화면의 비율 조정, 화면 숨기기 기능을 추가했다. 운동 종료 버튼을 누르면 진행한 운동 내역이 서버에 저장된다. 운동 기록 조회 페이지에서 사용자는 자신의 운동 기록 목록을 날짜별로 조회할 수 있으며, 운동 시간, 유사도 점수 등을 기준으로 필터링이 가능하다. 각 기록을 클릭하면 상세 페이지에서 타임라인별 점수를 확인할 수 있다.
코드 주요 설명	코드에 관한 주요 설명은 [붙임1]으로 작성하였다.
개발 내용	코드에 관한 주요 설명은 [붙임2]으로 작성하였다.
시연 영상	구글 폼으로 시연 영상을 제출하였다.
실행 파일 (선택)	구글 폼으로 .env 파일을 제출하였다.
기타 (선택)	서비스 배포 링크: (https://www.samadhi.kr/home)

붙임1 코드 주요 설명

1. MediaPipe 포즈 랜드마크 검출

MediaPipe의 Full 모델(pose_landmarker_full.task)을 로드하여 33개 전신 랜드마크를 추출하였다. 그리고 커스텀 훅 useMediaPipe()를 구현하여 영상 랜드마크와 웹캠 랜드마크를 구분하여 사용할 수 있도록 구성했다.

```
export function useMediaPipe() {
  const [isInitialized, setIsInitialized] = useState(false);
  const [error, setError] = useState<string | null>(null);

  const videoLandmarkerRef = useRef<PoseLandmarker | null>(null);
  const webcamLandmarkerRef = useRef<PoseLandmarker | null>(null);

  useEffect(() => {
    async function initMediaPipe() {
      try {
        const vision = await FilesetResolver.forVisionTasks(
          "https://cdn.jsdelivr.net/npm/@mediapipe/tasks-vision@latest/wasm"
        );

        const createLandmarker = () => {
          const options = {
            baseOptions: {
              modelAssetPath: "/models/pose_landmarker_full.task",
              delegate: "GPU" as "GPU" | "CPU",
            },
            numPoses: 1,
            minPoseDetectionConfidence: 0.5,
            minPosePresenceConfidence: 0.5,
            minTrackingConfidence: 0.5,
            outputSegmentationMasks: false,
            runningMode: "VIDEO" as "VIDEO",
          };
          return PoseLandmarker.createFromOptions(vision, options);
        };

        const [videoLandmarker, webcamLandmarker] = await Promise.all([
          createLandmarker(),
          createLandmarker(),
        ]);
```

2. 유사도 알고리즘

(1) 좌표 정규화

```
const anchor = getMidpoint(LHIP, RHIP);
const scale = l2norm(LSHO, RSHO);

//랜드마크에 대해 픽셀 단위로 변환
const data = landmarks.map((mark: Landmark) => {
  // if ((mark.visibility || 0) < MIN_VISIBILITY) {
  //   return [0, 0, 0];
  // }
  return [
    (mark.x * width - anchor.x) / scale,
    (mark.y * height - anchor.y) / scale,
    (mark.z * width - anchor.z) / scale,
  ];
});
```

이 코드는 포즈 데이터를 카메라 위치, 인체 크기, 거리 변화에 불변한 특징 벡터로 만드는 과정이다. 중심점을 인체 중심부(양 엉덩이 중점)로 맞추고, 신체 크기 편차를 어깨 폭으로 정규화함으로써, 사용자 간의 신체 차이나 카메라 거리 변화가 유사도 계산에 미치는 영향을 최소화한다.

(2) 관절 각도 계산

```
// 벡터 BA와 BC
const ba = { x: a.x - b.x, y: a.y - b.y, z: a.z - b.z };
const bc = { x: c.x - b.x, y: c.y - b.y, z: c.z - b.z };

// 내적
const dot = ba.x * bc.x + ba.y * bc.y + ba.z * bc.z;

// 벡터 크기
const magBA = Math.sqrt(ba.x ** 2 + ba.y ** 2 + ba.z ** 2);
const magBC = Math.sqrt(bc.x ** 2 + bc.y ** 2 + bc.z ** 2);

if (magBA === 0 || magBC === 0) {
  return null;
}

// 각도 계산
const cosAngle = dot / (magBA * magBC);
```

각도 계산은 포즈의 관절 형상 정보를 벡터 기하학적으로 정량화한 것이다. 세 점이 형성하는 벡터 간의 내적(dot product)을 이용하여 각도를 구하며 이는 인체의 특정 자세(예: 팔꿈치 굽힘, 무릎 굽힘)의 정량적 표현이 된다. 코드는 clip을 통해 수치 오차로 인한 cosθ의 범위 초과를 방지하며 결과는 도 단위(°)로 변환되어 직관적인 해석이 가능하다.

(3) 데드존 필터

```
/**
 * 데드존 필터 적용
 */
function applyDeadZone(
  key: keyof JointAngles,
  newAngle: number,
  previousAngles: Partial<JointAngles>,
  updatePreviousAngles: (key: keyof JointAngles, angle: number) => void
): number {
  const prevAngle = previousAngles[key];

  if (prevAngle === undefined || Object.keys(previousAngles).length === 0) {
    updatePreviousAngles(key, newAngle);
    return newAngle;
  }

  const diff = Math.abs(newAngle - prevAngle);

  // 변화가 데드존 이하면 이전 값 유지 (떨림 방지)
  if (diff < DEAD_ZONE) {
    return prevAngle;
  }

  // 변화가 크면 새 값으로 업데이트
  updatePreviousAngles(key, newAngle);
  return newAngle;
}
```

데드존 필터는 포즈 추정에서 흔히 발생하는 노이즈성 미세 떨림을 제거하기 위한 안정화 기법이다. 작은 각도 변화는 실제 동작보다는 인식 오차일 확률이 높기 때문에 2° 이하 변화는 무시하여 포즈의 연속성과 부드러움을 확보한다. 이를 통해 프레임 간 유사도 측정 시 불필요한 진동성 편차가 줄어든다.

(4) 코사인 유사도

```
// 1) 코사인 유사도 (클램프)
let cosine = dot / (mag1 * mag2);
if (cosine > 1) {
  cosine = 1;
} else if (cosine < -1) {
  cosine = -1;
}
const cosineScore = ((cosine + 1) / 2) * 100; // 코사인 점수 -> 방향
```

코사인 유사도는 두 포즈 벡터의 방향 일치도, 즉 자세의 형상적 유사성을 나타낸다. 값이 1에 가까울수록 두 포즈의 방향이 동일하며, -1에 가까울수록 반대 방향이다. 100점 척도로 변환한 산출물은 형상 유사도를 직관적으로 표현하며, 회전·스케일 변화에 불변하는 특징을 갖는다.

(5) 포즈 시퀀스 안정성

```
export function calcJitter(poseLandmarks: Landmark[][]) {
  const frameDiffs = [];

  for (let i = 1; i < poseLandmarks.length; i++) {
    const prev = poseLandmarks[i - 1];
    const curr = poseLandmarks[i];

    // 각 관절의 프레임 간 유클리드 거리
    const distances = curr.map((joint, j) => {
      const dx = joint.x - prev[j].x;
      const dy = joint.y - prev[j].y;
      const dz = joint.z - prev[j].z;
      return Math.sqrt(dx * dx + dy * dy + dz * dz);
    });

    frameDiffs.push(distances);
  }

  // 관절별 jitter (표준편차)
  const joints = frameDiffs[0].length;
  const jitterPerJoint = Array(joints).fill(0);
```

이 지표는 프레임 간 관절의 흔들림을 정량적으로 평가하며, 값이 낮을수록 포즈 추정이 안정적임을 의미한다. 전처리(정규화) 전후의 jitter 값을 비교하면 전처리의 안정화 효과를 확인할 수 있다.

(6) 전체 파이프라인

```
export const CalculateCosAndEuc = (P1: number[], P2: number[], ) : CosAndEuc | null =>  {
    const n = P1.length;

  if (n !== P2.length || n === 0) {
    return null;
  }

  // 내적과 노름
  let dot = 0;
  let sum1 = 0;
  let sum2 = 0;
  let diffSum = 0;

  for (let i = 0; i < n; i++) {
    const a = P1[i];
    const b = P2[i];

    dot += a * b;
    sum1 += a * a;
    sum2 += b * b;
    const d = a - b;
    diffSum += d * d;
  }
```

전체 파이프라인은 MediaPipe 랜드마크를 입력받아 (1)좌표 정규화 → (2)벡터화 → (3) 코사인 점수 산출 순으로 진행된다. 최종적으로 출력되는 S∈[0,100]은 두 포즈의 정량적 유사도를 나타내며, 동작 평가, 자세 비교, 실시간 피드백 등 다양한 응용에 활용 가능하다.

3. 자세 분류 알고리즘

전체 관절 랜드마크 및 각 관절 각도 간 코사인 점수를 기반으로 일정 임계값 이상일 경우 특정 자세를 만족하는 것으로 판단하여 분류를 진행한다.
영상 및 웹캠을 실시간으로 각 프레임마다 자세를 분류하며, 이를 이용하여 한 자세를 유지할 경우 자세의 시작 시간, 끝 시간, 해당 자세의 사용자와 영상 강의자 간의 유사도를 제공한다. 계산된 자세별 유사도 기록은 운동 종료 시 서버에 전송되어 기록 페이지에서 조회할 수 있다.

```
export function classifyPoseWithVectorized(vectorized: number[], angles?: JointAngles) {
    let bestPose = "unknown";
    let maxDistance = 0;
    const distPerPose: Record<string, number[]> = {};
    const VEC_THRESHOLD = 88;
    const ANGLE_THRESHOLD = 96;

    // 좌우 반전 버전 생성
    const mirroredVectorized = normalizeMirroredVectorized(vectorized);
    const mirroredAngles = angles ? normalizeMirroredAngles(angles) : angles;

    for (const [name, poseVectorized] of Object.entries(poseVectorizedData)) {
        const poseAngles = poseData[name as keyof typeof poseData];
      const calcVecDistance = (a: number[]) => {
        const similarity = CalculateSimilarity(poseVectorized, a, 1);

        // 100에 가까울 수록 유사, 0에 가까울수록 다름
        return similarity;
      };

      const calcDistance = (a: JointAngles) => {
        const keys = Object.keys(a) as (keyof JointAngles)[];

        // dot product, magnitude of a and poseAngles
        const { dot, magA, magB } = keys.reduce(
          (acc, key) => {
            const valA = a[key];
            const valB = poseAngles[key];
            acc.dot += valA * valB;
            acc.magA += valA * valA;
            acc.magB += valB * valB;
            return acc;
          },
          { dot: 0, magA: 0, magB: 0 }
        );

        const similarity = dot / (Math.sqrt(magA) * Math.sqrt(magB));

        // 100에 가까울 수록 유사, 0에 가까울수록 다름
        return similarity * 100;
      };

      // 원본과 반전 둘 다 계산
      const distanceOriginal = calcVecDistance(vectorized);
      const distanceMirrored = calcVecDistance(mirroredVectorized);

      const angleDistanceOriginal = angles && poseAngles ? calcDistance(angles) : 0;
        const angleDistanceMirrored = mirroredAngles && poseAngles ? calcDistance(mirroredAngles) : 0;

      // 더 유사한 쪽 선택
      const maxForThisPose = distanceOriginal > distanceMirrored ? distanceOriginal : distanceMirrored;
      const _angleMaxForThisPose = distanceOriginal > distanceMirrored ? angleDistanceOriginal : angleDistanceMirrored;
      const angleMaxForThisPose = angles ? _angleMaxForThisPose : ANGLE_THRESHOLD + 1; // 각도 데이터가 없으면 각도 조건 무시

      if (maxForThisPose > VEC_THRESHOLD && angleMaxForThisPose > ANGLE_THRESHOLD && maxForThisPose > maxDistance) {
        maxDistance = maxForThisPose + angleMaxForThisPose;
        bestPose = name;
      }

      distPerPose[name] = [maxForThisPose, angleMaxForThisPose];
    }
```

```
useEffect(() => {
  console.log(video.poseClass, currentPose);
  if (video.poseClass !== currentPose) {
    const now = Date.now();
    if (currentPose !== "unknown") {
      // End the previous pose timeline
      setTimelines((prev) => {
        const updated = [...prev];
        const lastTimeline = updated[updated.length - 1];
        const avgSimilarity =
          similarities.length > 0
            ? similarities.reduce((a, b) => a + b, 0) / similarities.length
            : 0;
        if (lastTimeline && lastTimeline.endTime === 0) {
          lastTimeline.endTime = now;
          lastTimeline.similarity = avgSimilarity;
          setSimilarities([]); // Reset similarities for the next pose
        }

        return updated;
      });
    }
    // Start a new pose timeline
    if (video.poseClass !== "unknown") {
      setTimelines((prev) => [
        ...prev,
        { pose: video.poseClass, startTime: now, endTime: 0, similarity: 0 },
      ]);
      setCurrentPose(video.poseClass);
    }
  }
}, [video.poseClass]);
```

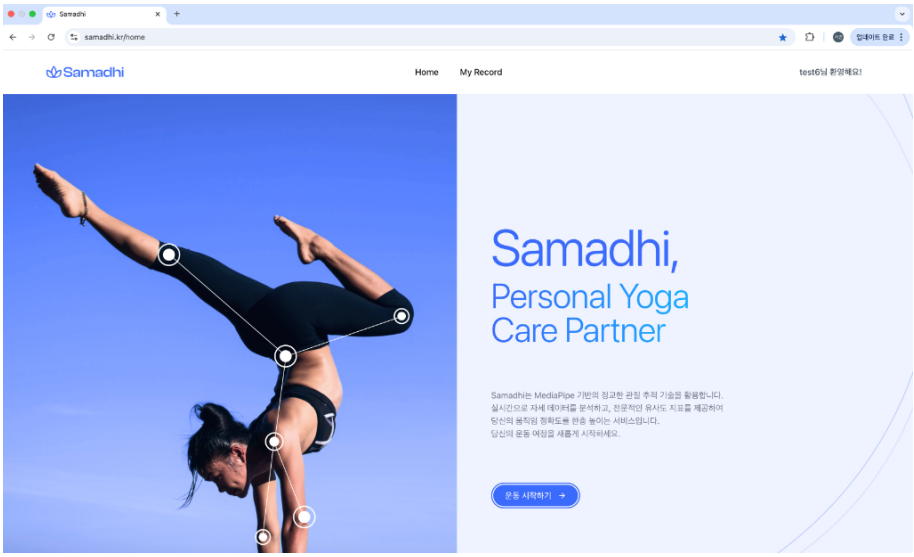
붙임2 개발 내용

- 개발 내용 요약

- 영상/웹캠 각각의 포즈 분류 코드 작성
- 영상/웹캠 간의 유사도 계산 코드 작성
- 영상 자세를 기반으로 영상 클리핑 & 각 자세 시퀀스별 평균 유사도 계산
- 자세 분류를 위한 정답 자세 데이터 추가
- 영상별 운동기록 저장/조회 코드 작성
- 사용자 편의를 위한 샘플 비디오 업로드
- JWT 토큰을 이용한 로그인 기능 구현
- 영상/웹캠 간 유사도 및 사용자 운동 기록 시각화
- 사용자 운동 타임라인 시각화
- FE, BE AWS에 배포
- 샘플 비디오, 프로필 사진, 썸네일과 같은 정적파일 S3에 업로드
- DB 구축
- github action을 이용한 배포 자동화

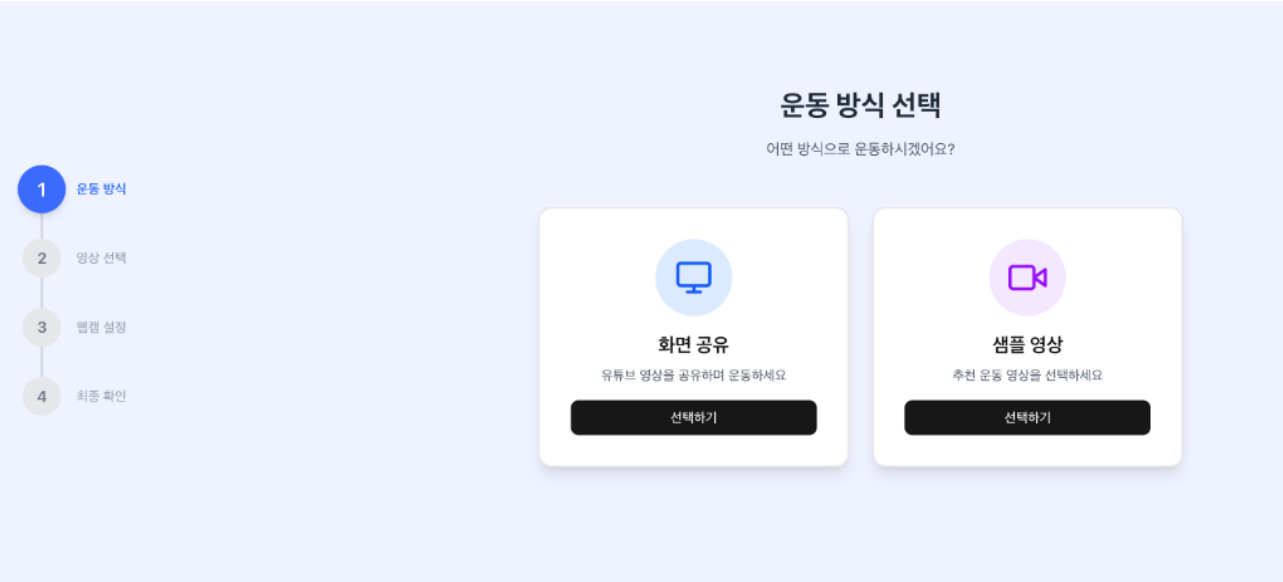
- 실제 서비스 화면

우선 홈화면과 로그인, 회원가입 기능을 구현했다. 홈화면에서 '운동 시작하기' 버튼을 누르면 운동 준비 프로세스가 시작된다.

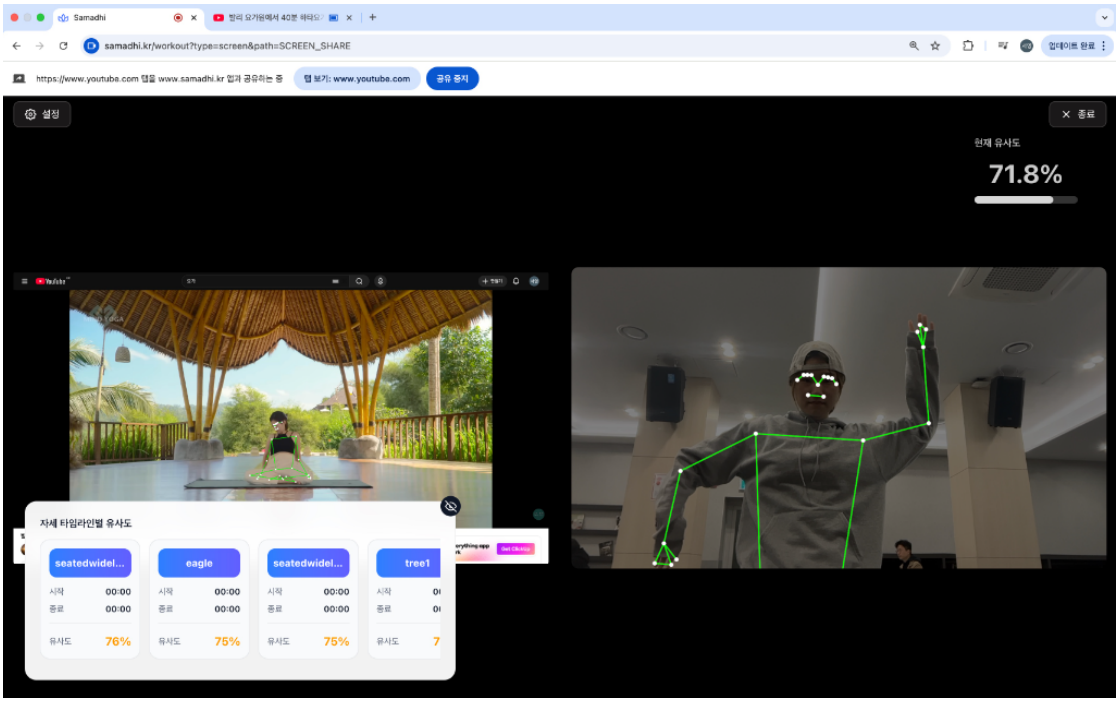


운동 준비 프로세스는 4단계로 이루어진다. 먼저 샘플 영상 모드 또는 화면 공유 모드 중 하나를 선택한다. 샘플 영상 모드는 시스템에서 제공하는 전문가의 요가 동작 영상을 활용한 운동 모드이고, 화면 공유 모드는 사용자가 선호하는 영상을 틀고 화면 공유를 활용해서 운동이 가능한 모드이다. 이를 통해 다양한 요가 콘텐츠 활용이 가능하

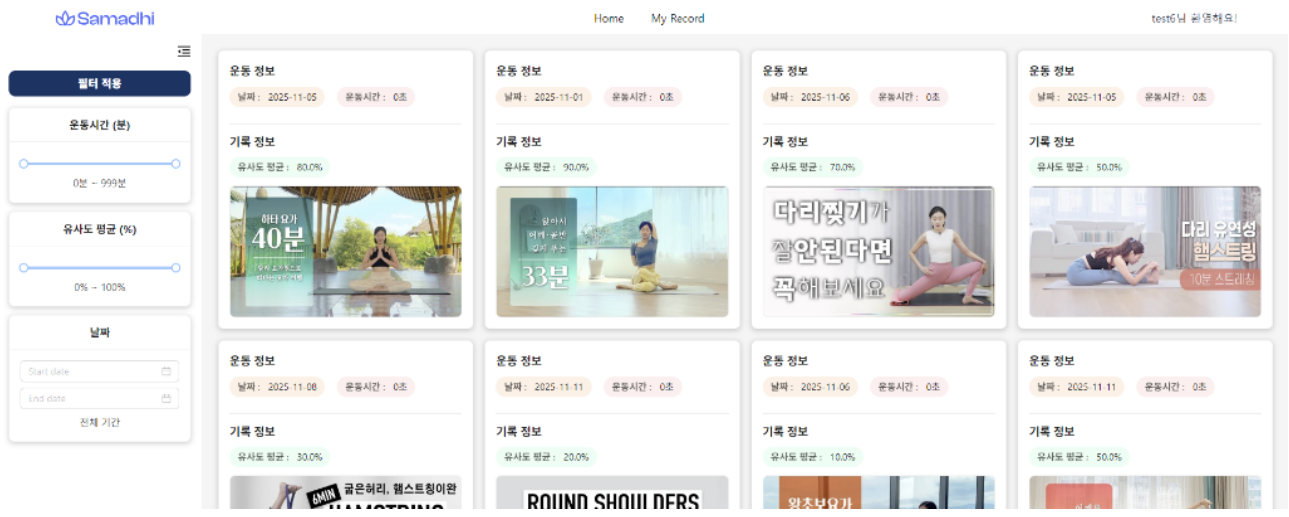
여 사용자의 선택권을 확대하고자 했다. 그 후 웹캠 설정이 이루어진다. 브라우저에서 웹캠 접근 권한을 허용하면, 웹캠 미리보기가 뜨게 된다. 최종적으로 선택한 영상과 웹캠 설정을 최종 확인하고 준비가 완료되면 운동 페이지로 이동하게 된다.



운동 진행 중 화면의 경우, 화면을 좌우로 분할하여 왼쪽에는 참고 영상, 오른쪽에는 사용자의 웹캠 화면을 표시한다. 실시간 유사도 점수를 상단에 표시하여 즉각적인 피드백을 제공하고, 왼쪽 하단에는 참고 영상에서 분류된 자세를 바탕으로 자세 타임라인별 유사도가 표시된다. 운동 중 인식된 자세가 변경될 때마다 자동으로 타임라인 구간을 생성하고 각 구간은 자세 이름, 시작/종료 시간, 해당 자세에 대한 평균 유사도 점수를 포함한다. 또한 사용자 편의성을 위해 설정에서 각 화면의 비율 조정, 화면 숨기기 기능을 추가했다.



운동 종료 버튼을 누르면 진행한 운동 내역이 서버에 저장된다. 운동 기록 조회 페이지에서 사용자는 자신의 운동 기록 목록을 날짜별로 조회할 수 있으며, 운동 시간, 유사도 점수 등을 기준으로 필터링이 가능하다. 각 기록을 클릭하면 상세 페이지에서 타임라인별 점수를 확인할 수 있다.



- CI/CD 관련

CICD 파이프라인 구축을 위해 깃허브 actions를 사용하였다. main 브랜치에 push 이벤트가 발생하면 frontend와 backend 폴더 중 변경이 발생한 영역에 대해 deploy-frontend 혹은 deploy 워크플로우가 작동한다. 워크플로우는 다음과 같은 과정으로 동작한다. ecr private 레지스트리에 이미지를 push하기 위한 aws 로그인 과정을 거친 후 docker 이미지를 생성한다. docker 이미지 빌드가 완료되면 ecr에 이미지를 push하고 ec2 서버의 ssh 환경에 접속하여, docker compose를 통해 ecr에 올라간 이미지를 실행시킨다. 다만, 프론트엔드의 경우 docker compose를 돌릴 때 환경변수를 전달해주는 것이 아니라 이미지 빌드 시에 전달해주어야 하기 때문에 워크플로우의 docker build 명령어에 환경변수를 전달해주었다.