Cheating Detection Final Report

임정민 Author[1], 하인혜 Author[2], 이강복 Author[3]

1 INTRODUCTION

Covid-19 사태 이후로 우리의 생활환경이나 제도에 큰 변화가 찾아왔다. 그 중 학생에게 가장 큰 변화는 비 대면 수업과 비 대면 시험일 것이다. 대면과 비 대면시험을 결정할 수 있는 결정권자들은 각각의 방법이 가진 문제점 때문에 혼란을 겪기도 한다. 그 이유는 비 대면 시험의 불공정성 때문인데, 대면 시험을 치르게 되면 공정성의 문제는 어느정도 해결할 수 있지만 방역 문제가 생긴다. 학교에서 방역 수칙을 철저히 지켜도 대중교통, 불필요한 외부 접촉이 발생하고 이는 전염병 확산의 우려가 생긴다. 그리고 공정성의 문제도 여전히 존재하는데, 전염병에 걸리거나 혹은 감염되지 않더라도 자가격리를 해야 하는 상황에 시험이 겹치면 시험을 치르지 못하게 되고, 이로 인해서 시험의 공정성에 타격이 생길 수 있다. 그래서 비 대면 시험으로 치르면 부정행위를 막을 제도가 현저히 부족하고 이로 인해 양심적으로 시험을 보는 사람이 피해를 보는 상황이 연출된다. 따라서 비 대면 시험의 공정성을 확립하고 코로나 사태가 끝날 때까지, 혹은 이후에 생길 수 있는 비슷한 상황에 대비하여 제도를 확립하는데 목적이 있다.

비 대면 시험의 부정행위는 어떤 것들이 있을까? 여러가지의 창의적인 방법이 있지만 대표적인 것은 카메라 사각지대를 이용한 부정행위, 대리시험이 있다. 이를 우리는 Eye Tracking Model, Self-certification Model 을 통해 해결하고 웹사이트에 연동하여 비 대면 시험의 공정성을 확립한다.

2 RELATED WORK

2.1 Substitute Examination

그렇다면 기존에 만들어진 제품들은 어떻게 다음과 같은 치명적인 부정행위를 막으려고 시도했는지, 그리고 그것의 한계는 무엇이며, 어떻게 해결할 것인지도 포함한다.

Figure 1. Ontest & Monito's Anti-Substitute Examination Strategies

2.1.1 OnTest

OnTest에서는 대리 시험을 방지하기 위해서 시험 응시자의 신분증 사진을 촬영해서 그것의 신분증 사진과 실제 응시자의 사진을 시험 감독관이 비교하는 방식이다. 이는 대면 시험에서 진행하는 방식과 유사해서 큰 문제가 없다고 판단할 수도 있지만 온라인 시험 특징 상 많은 응시자가 발생하기 때문에 관리 감독자가 번거로움을 겪을 수 있다.

2.1.2 Monito

마찬가지로 모니토의 제품도 신분증의 사진을 촬영하는 방식으로 진행한다. 따라서 위와 같은 문제점이 동일하게 발생한다.

2.1.3 How to Solve?

따라서 우리는 이 문제를 해결하기 위해서 Self-certification Model을 생성한다. 이는 신분증에 있는 사진과 시험 응시자를 자동으로 비교하고, 더 나아가서 Anti-spoofing Tech를 이용하여 대리시험을 더욱 효과적으로 막을 수 있게 한다.

2.2 Cheating using Camera Blind Spots

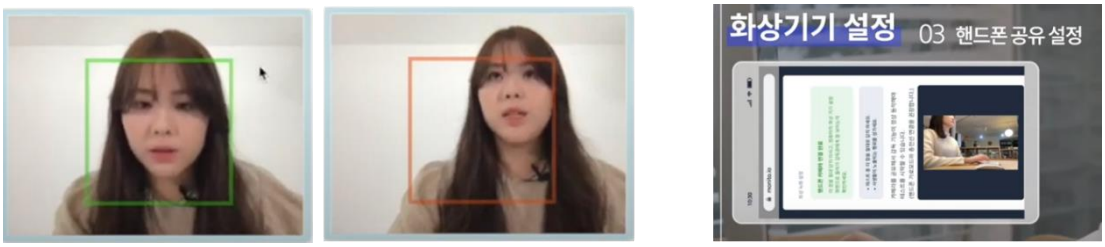


Figure 2. Ontest & Monito's Anti-Cheating using camera blind spots

2.2.1 OnTest

OnTest에서는 카메라 사각지대를 이용한 부정행위를 막기 위해서 사람을 인식한 뒤 그 사람이 특정한 사각형을 벗어나면 알람이 가는 형태로 구현하였다. 그러나 이는 사각형에서 시선을 움직여서 사각지대를 볼 수 있고 근본적인 해결책이라고 볼 수 없다.

2.2.2 Monito

Monito에서는 휴대폰 카메라와 웹 캠 두가지를 모두 사용하여 사각지대를 줄이려는 노력을 하였다. 그러나 이것은 사각지대가 줄어들기는 하지만 사각지대가 사라지는 것은 아니고 여전히 사각지대가 존재할 수 밖에 없으므로 이도 명확한 해결책은 아니다.

2.2.3 *How to Solve?*

따라서 우리는 이 문제를 해결하기 위해서 Eye Tracking Model을 사용한다. 이는 시험 응시자의 눈을 직접적으로 추적하고, 모니터를 제외한 곳으로 시선을 지속적으로 응시하면 알람을 보내는 모델이다.

3 METHOD

3.1 User-motion and Eye-Tracking

3.1.1 Camera Calibration

A. Introduction

일반적인 Pinhole Camera(e.g. 노트북의 내장 카메라)는 이미지에 상당한 왜곡을 일으킨다. 왜곡의 종류에는 방사형 왜곡(Radial Distortion)과 접선 왜곡(Tangential distortion)이 존재한다. 방사형 왜곡은 직선을 곡선으로 나타내게 한다. 이 왜곡은 이미지의 중심에서 멀리 떨어져 있을수록 심해진다. 예를 들어 아래 그림 n 의 경우, 체스 보드의 모서리가 빨간색 선으로 나타나 있는데, 실제 이미지는 빨간색 직선과 일치하지 않는, 즉 직선이 아닌 곡선으로 나타나 있는 것을 확인할 수 있다.

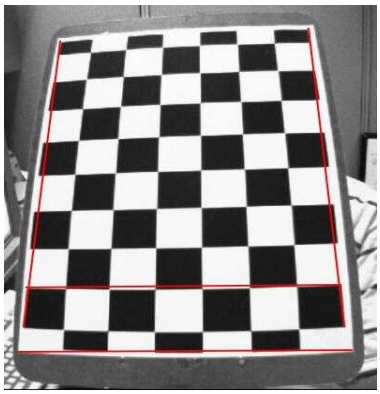


Figure 3. Image Distortion

방사형 왜곡은 다음과 같이 나타낼 수 있다.

$$r = x^2 + y^2$$

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

비슷하게, 접선 왜곡은 이미지 촬영 렌즈가 이미지 평면에 완벽하게 평행하게 정렬되지 않기 때문에 발생하며, 이로 인해 이미지의 일부 영역은 예상보다 더 가까이 보이는 현상이 발생한다. 접선 왜곡은 다음과 같이 나타낼 수 있다.

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$

$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

요약하면, 왜곡 계수(Distortion coefficients)라고 불리는 다섯가지의 매개변수를 구해야 왜곡된 이미지에 대한 보정이 가능하다.

$$Distortion\ coefficients = (k_1\ \ k_2\ \ p_1\ \ p_2\ \ k_3)$$

추가적으로, 일반적인 Pinhole 카메라(e.g. 노트북에 내장되어 있는 자그마한 카메라)는 실제 공간인 3 차원에서 카메라의 2 차원 이미지로 변환하는 과정에서 카메라의 위치나 방향 등의 외부 요인(External Parameter)이나 카메라 렌즈 와 이미지 센서의 거리인 초점거리(focal length, (f_x, f_y)), 광학축이 이미지 센서와 만나는 점(i.e. 카메라 렌즈의 중심에서 이미지 센서에 내린 수선의 발)의 좌표인 주점(principal point, (c_x, c_y)), 이미지 센서의 cell array 의 y 축이 기울어진 정도인 비대칭 계수(skew_coefficient, 일반적으로 고려하지 않음, 0)에 의해 왜곡이 발생하게 된다. 이들을 카메라 매트릭스(Camera Matrix)라 부르며,이것을 구하게 되면 발생한 왜곡을 보정할 수 있다는 의미가 된다.

$$Camera\ Matrix\ = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

외부 요인은 3 차원 좌표를 이미지 좌표로 변환하는 회전 및 이동 벡터이다. 이후 Head Pose Estimation 를 구하기 전에 필요한 값이다.

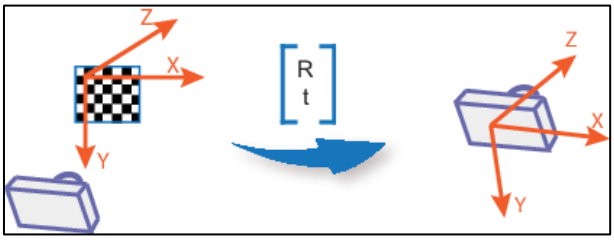


Figure 4. External Parameter

$$External\ Parameter = [R|t]\ = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix}$$

결론적으로 Camera Calibrate 과정에서는 이후 과정에서 카메라로 받아온 이미지의 왜곡을 보정하기 위해 위에서 언급한 파라미터들을 구하는 과정이다.

B. Process

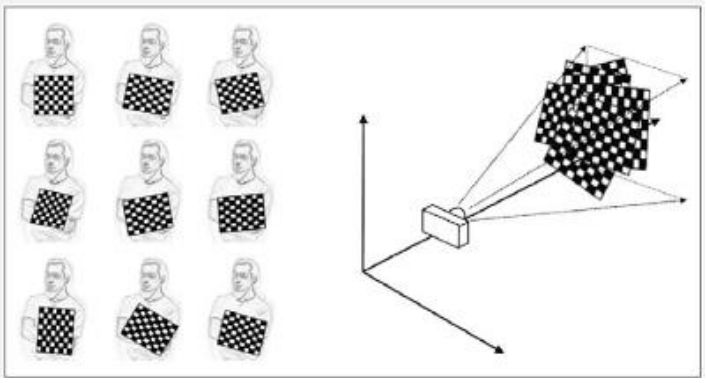


Figure 5. Camera Calibration

파라미터를 구하기 위해, 해야 할 일은 잘 정의된 패턴이 담긴 샘플 이미지(e.g. 체스판)를 제공하는 것이다. OpenCV 라이브러리를 통하여 체스판 이미지 내부에 존재하는 체스판의 사각형 모서리를 찾고 찾은 모서리를 통해 이미지에서의 좌표를 구하고, 실제 좌표는 알고 있으므로 이 두가지 좌표를 통해 파라미터를 구할 수 있게 된다. 카메라 보정을 잘 나오게 하려면 최소 10 개 패턴의 이미지가 필요하다고 알려져 있으며, 현재 우리의 프로젝트의 demo 버전은 신속한 테스트를 위해 8 개의 이미지만을 사용하고 있으나 이 패턴 이미지의 수를 늘리는 것은 간단하기 때문에 더 다양한 패턴 이미지가 들어갈 수 있도록 늘릴 예정이다.

카메라 보정을 위해 필요한 데이터는 체스보드판이 담긴 이미지, 3 차원 상에 존재하는 실제의 지점에 대한 집합과 2 차원 이미지 상에서의 지점 집합이 필요하다. 2 차원 이미지 상에서의 포인트는 체스보드에서 두개의 검은 정사각형이 서로 맞닿아 있는 위치이기 때문에 단순하게 구할 수 있다. 하지만 실제 체스판의 좌표는 정적인 카메라에서 찍히고 체스판은 다른 위치와 방향에 배치가 되기 때문에 (X, Y, Z)의 값을 알아야 한다. 하지만 단순화하기 위해 우리는 체스판은 XY 평면에 정지 해 있고(i.e. Z=0) 이에 따라 카메라가 이동되었다고 가정할 수 있다. 이러한 가정에 따라 실제 좌표는 Z=0 인 상태에서 (x, y) = (0, 0), (1,0), ...으로 점들을 전달할 수 있게 된다. 여기서 실제 좌표를 object points, 이미지 좌표를 image points 라고 한다.

Image points 는 OpenCV 에 존재하는 cv2.findChessboardCorners() 함수를 통해 구할 수 있으며, 찾은 결과를 이미지로 표현하면 그림 6 과 같다.

Figure 6. Results of cv2.findChessboardCorners()

Object points 와 Image Points, 그리고 그 때의 frame 을 input 으로 하여 OpenCV 에 존재하는 cv2.calibrateCamera() 함수를 통하여 Camera Matrix, Distortion coefficients, rotation and translation vector 를 구할 수 있다.

3.1.2 Head Pose Estimation

머리 방향을 추정하는 것은 사용자가 고개를 돌려 부정행위를 하는 것을 방지할 수 있는 방법이다. 이 추정은 Translation과 Rotation을 통해 머리의 방향을 추정할 수 있다. 2차원 이미지를 통해서 물체의 자세를 추정할 수 있는 방법은 바로 PnP(Perspective-n-Point)라는 방법을 이용하여 해결할 수 있다.

PnP 문제의 식은 다음과 같다.

$$s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

이전 단락에서 보았던 카메라 내부와 외부 파라미터의 형식이 포함되어 있다는 것을 알 수 있다. 다시 말해서 이 식을 해결하기 위해서 우리가 알아야 할 값은 이미지 공간에서의 2d좌표, 실제 공간에서의 3d좌표, 그리고 카메라 내부 파라미터를 알아야 한다.

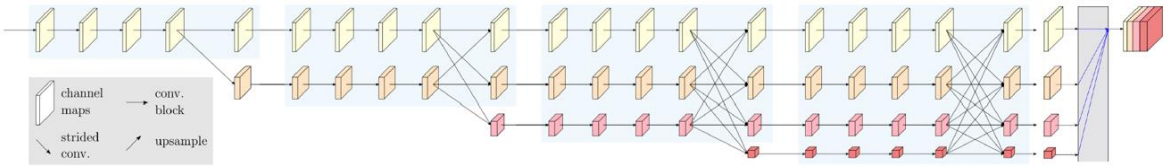


Figure 7. HRNet architecture

사람의 얼굴이 포함된 이미지를 받고, 해결해야 할 문제는 키 포인트 좌표를 찾아야 한다. 다시 말해서 얼굴의 landmark를 감지하는 것이다. 이 부분에서는 사전 학습된 HRNet_facial_landmark_detection을 통하여 얼굴 탐지와 landmark 탐지를 실시하였다. HRNet의 작동 방식을 간단하게 요약하면, 대부분 CNN Architecture들의 경우는 High Resolution에서 Low Resolution으로 점점 해상도를 줄여나가는 방식인데, 이 모델의 경우는 High Resolution을 유지한 채로 병렬적으로 Low Resolution을 적용하는 방식이라고 보면 된다. Landmark를 탐지한 이후 구한 모든 값을 사용하는 것이 아니라, 본 프로젝트에서는 nose ridge, nose base, left-eye corners, right-eye corners를 사용한다. 이후 2d좌표에는 x와 y축 좌표만, 3d 좌표는 모든 축의 좌표를 사용해 이미지 좌표와 실제 공간에서의 좌표를 구하고, 이전 과정에서 구한 카메라 내부 파라미터를 이용하여 OpenCV에 존재하는 cv2.solvePnP()함수를 적용한다.

cv2.solvePnP() 함수를 이용하면 translational vector와 rotational vector의 형태로 반환이 되어 나오는데, 이것은 우리가 원하는 Matrix형태가 아닌 Rodrigues 형태이다. 따라서 우리가 원하는 Matrix형태로 바꾸기 위해 구해진 값에 cv2.Rodrigues()함수를 적용하여 원하는 행렬식의 형태로 얻는다.

사실 마지막으로, 사용자가 얼굴을 얼마만큼 돌렸는지 확인하기 위해서는 단순히 회전 행렬식이 아닌 어느 축을 기준으로 얼마만큼의 각도로 돌렸는지 확인해야 한다. 이를 구하기 위해서는 회전 행렬식의 생성이 된 배경을 알아보아야 한다.

3d 물체는 3개의 직교 축으로 회전이 가능하고, 이를 yaw, pitch, roll 이라고 부른다.

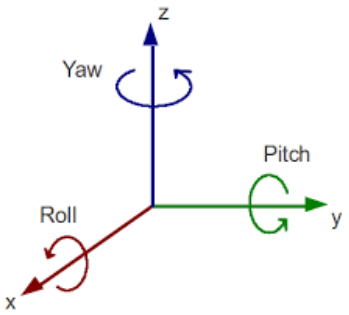


Figure 8. Yaw, Pitch, Roll

Yaw는 z축을 중심으로 회전하는 것을 의미하며, 사람이 얼굴을 좌우로 도리도리 움직일 때 각도가 변한다.

$$R_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Pitch는 y축을 중심으로 회전하는 것을 의미하며, 사람이 얼굴을 앞뒤로 끄덕끄덕 움직일 때 각도가 변한다.

$$R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}$$

Roll은 x축을 중심으로 회전하는 것을 의미하며, 사람이 얼굴을 좌우로 갸우뚱 움직일 때 각도가 변한다.

$$R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix}$$

회전 행렬식은 이러한 행렬들의 곱으로 이루어져 있으며, 그 식은 다음과 같다.

$$\begin{aligned} R(\alpha,\beta,\gamma) &= R_z(\alpha)R_y(\beta)R_x(\gamma) \\ &= \begin{pmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{pmatrix} \\ &= \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \end{aligned}$$

회전 행렬식을 이용해 사용자가 얼굴이 어느 축을 기준으로 얼마나 돌아 갔는지에 대한 정보를 가지고 있는 α, β, γ값을 구하는 방법은 삼각함수의 성질을 이용하여 구할 수 있다.

$$\alpha = \tan^{-1}(\frac{r_{21}}{r_{11}})$$
$$\beta = -\sin^{-1}(r_{31})$$
$$\gamma = \tan^{-1}(\frac{r_{32}}{r_{33}})$$

3.1.3 *Eye-Tracking*

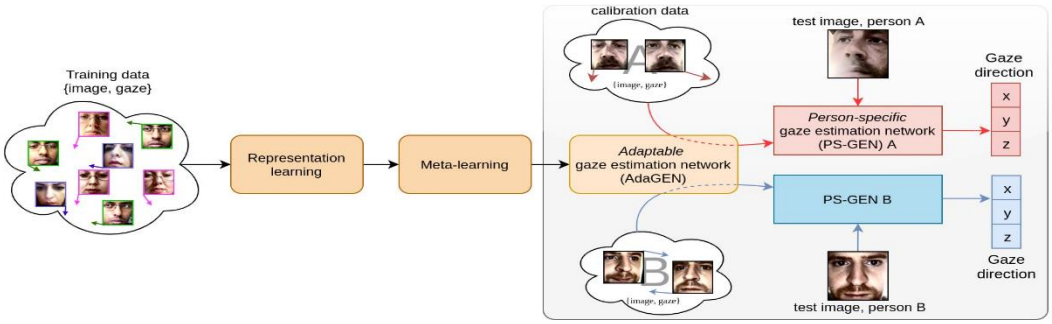


Figure 9. FAZE architecture

본 프로젝트에서 사용되는 Eye-Tracking Model은 바로 FAZE(Few-shot Adaptive Gaze Estimation)이다. FAZE의 첫 번째 단계는 person-specific 측면을 포함하여 시선 방향에 관련된 정보를 인코딩하는 모델을 학습을 하고, Meta-Learning의 방법으로 few-shot person-specific gaze estimator를 학습하고, 이를 통해 Adaptable Gaze Estimation Network(AdaGEN)을 생산하게 된다. 그리고 이후에는 사전 학습된 모델을 통하여 특정 사용자에 대한 시선 추적 모델을 만들게 된다.

A. *Representation Learning (Disentangling Transforming Encoder-Decoder)*

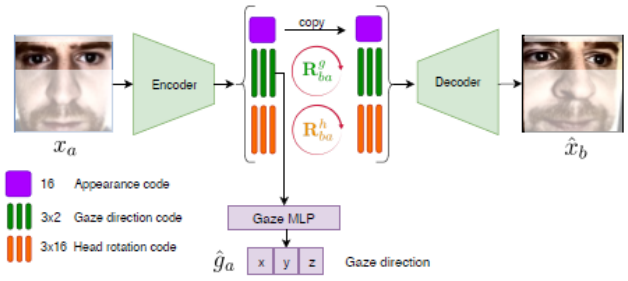


Figure 10. DT-ED architecture

첫 번째 단계에서 Input 은 한 쌍의 사람의 이미지 (x_a x_b) 가 된다. 이 모델은 이미지로부터 세 가지의 요인들을 구별해준다. 이미지에서의 눈의 위치인 Appearance, 시선의 방향, 머리의 방향을 의미하고 이것은 다음과 같이 표현될 수 있다.

$$z = \{z^a; z^g; z^h\}$$

우리는 x_a 에서 x_b로 이동하는 시선의 방향의 변화를 설명하기 위해 $R_{ba}^{g} = R_{b}^{g}(R_{a}^{g})^{-1}$를 계산할 수 있고, 이 회전 행렬은 머리의 회전에도 적용시킬 수 있다. 이것을 구하는 방법은 입력 샘플의 쌍에 대한 시선 및 머리 방향에 대한 실제 레이블을 이용해서 구할 수 있다.

$$R^{(\theta,\emptyset)} = \begin{bmatrix} \cos\emptyset & 0 & \sin\emptyset \\ 0 & 1 & 0 \\ -\sin\emptyset & 0 & \cos\emptyset \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

이후, 회전시킨 머리방향, 시선의 방향, 그리고 기존의 Appearance 를 이용하여 x_b를 Decoder Function 을 통해 새로 재구축 할 수 있다.

이러한 과정을 거치는 encoder-decoder 모델을 다음과 같은 loss function 을 사용하여 훈련을 한다.

$$\mathcal{L}_{full} = \lambda_{recon}\mathcal{L}_{recon} + \lambda_{EC}\mathcal{L}_{EC} + \lambda_{gaze}\mathcal{L}_{gaze}$$

Loss function 은 크게 Reconstruction, Embedding consistency, Gaze Direction Loss 세가지로 나눌 수 있고, 상수 값인 λ의 경우, $\lambda_{recon} = 1, \lambda_{EC} = 2, \lambda_{gaze} = 0.1$ 로 설정하였다.

먼저 Reconstruction Loss 의 경우에는, 간단한 l_1 reconstruction loss 를 사용하며, Input image x_b와새로 구축한 이미 지인 $\hat{x}_b$에 대해서 Reconstruction Loss 는 다음과 같다.

$$\mathcal{L}_{recon}(x_b, \hat{x}_b) = \frac{1}{|x_b|}\sum_{u\in x_b, \hat{u}\in\hat{x}_b} |\hat{u} - u|$$

두번째 경우인 EC(Embedding Consistency) Loss 는, 모델을 구현한 논문에서 새롭게 정의한 단어인데, 이것은 Encoder Network 는 항상 사람의 이미지를 외형은 다르지만 동일한 시선 방향에 대해서는 유사한 features 로 embed 한다는 의미이다. 이것은 시선의 방향만 다른 한 쌍의 이미지를 필요로 하고 잠재적인 시선 특징들 사이의 Consistency 를 계산하기 이전에, 실제 시선의 방향인 g_a를 이용하여 회전 행렬인 R_a^g를 구하고, 이 행렬의 역행렬을 적용하여 시선 특징들을 정면화를 한다. EC 적용시에, 모든 시선 특징을 사람들 사이에 유사하도록 순진하게 적용하는 것은 다른 사람 사이에서 임베딩 결과가 나와야 하는 것을 무시할 수 있기 때문에 EC 를 동일한 사람의 이미지쌍에서만 적용을 한다. 배치 당 B 개의 이미지 샘플을 고려한다고 할 때, 다음과 같은 공식으로 EC Loss 를 계산할 수 있다.

$$f(x) = (R^g)^{-1}x$$

$$\mathcal{L}_{EC} = \frac{1}{B}\sum_{i=0}^{B} \max_{\substack{j=1\dots B \\ id(i)=id(j)}} d(f(z_i^g), f(z_j^g))$$

$f(x)$ 는 잠재적인 시선 특징들을 정면화 시키는 함수이며, $id(i)$ 함수는 사람의 i 번째 샘플의 사람의 identity 를 반환하는 함수이고, d는 3D vectors 사이에서의 열 우선방식의 평균

Angular distance 를 의미하고, Max 함수는 가장 멀리 떨어져 있는 사람 내 특징들 간의 차이를 최소화하는 역할을 한다.

마지막으로, Gaze Direction Loss 의 경우는, 간단한 MLP(Multi-Layer-Perceptron)에 의해 파라미터화 된 $\mathcal{G}: z^g \rightarrow \hat{g}$ 를 통하여 시선 추정의 추가적인 목적함수를 설정한다. Gaze Direction Loss 는 다음 수식에 의해 계산이 된다.

$$\mathcal{L}_{gaze}(\hat{g}, g) = \cos^{-1}\left(\frac{\hat{g} \cdot g}{\|\hat{g}\|\|g\|}\right)$$

B. *Adaptable Gaze Estimator (AdaGEN)*

한편, 이 시선 추적 모델은 가능한 적은 데이터를 이용하여 모델을 만드는 것이 목적인데, 단순하게 훈련 데이터를 이용하여 학습을 하고 사용 가능한 샘플만을 이용하여 fine-tuning 하는 방법이 있으나, 소수의 샘플만 사용 가능한 실제 환경에서 이러한 접근 방식은 Overfitting 의 문제로 이어질 수 있기 때문에 이 문제를 완화하기 위해 MAML(Model-agnostic meta-learning)이라는 학습 방법을 제안하였다.

MAML 의 핵심 키워드는 'Model Agnostic'과 'Fast adaptation'이다. Model Agnostic 은 모델에 상관없이 gradient descent 방식을 사용하는 모든 모델에 이 방식을 적용할 수 있다는 의미이고, Fast adaptation 은 새로운 task 를 빠르게(i.e. 적은 Update) 학습할 수 있다는 의미로, 메타러닝의 핵심 개념에 해당한다.

시선 추적에서, MAML 의 목적은 시선 추적 모델 $\mathcal{M}$의 초기 가중치 θ^*를 학습하여 어떤 사람 $\mathcal{P}$의 적은 수의 교정된 이미지를 이용하여 fine-tune 이 된 모델 $\mathcal{M}_{\theta^*}$가 아직 학습되지 않은 그 사람의 검증 이미지 샘플에 대해서 잘 일반화가 되는 것이다. 다시 말해, $\mathcal{M}_{\theta^*}$가 우리가 학습하고자 하는 높은 적응력이 있는 시선 추적 모델인 AdaGEN 을 형성하는 것이다.

MAML 에서 θ^*를 학습하는 전반적인 과정은 다음과 같다. 먼저 전체 사람의 데이터셋 S를 meta-training(S^{train})과 meat-testing(S^{test})으로 겹치지 않게 나눈다. 각 meta-training n번째 iteration 에서, S^{train} 으로부터 한 사람의 데이터셋인 $\mathcal{P}^{train}$ 을 랜덤하게 고른다. 이 때 $\mathcal{P}^{train} = \{\mathcal{D}_c^{train}, \mathcal{D}_v^{train}\}$ 은 k 개의 training 이미지로 구성된 calibration set 인 $\mathcal{D}_c^{train} = \{(z_i^g, g_i) | i = 1 \dots k\}$ 와 랜덤 샘플링을 통하여 만들어진 또다른 l 개의 이미지로 구성된 validation set 인 $\mathcal{D}_v^{train} = \{z_j^g, g_j | j = 1 \dots l\}$로 이루어져 있다.$(k, l \leq 20)$

Meta-optimization 의 첫번째 과정은 calibration set 인 $\mathcal{D}_c^{train}$의 loss 를 계산하고 n번째 gradient step 에서의 θ_n을 업데이트 한다.

$$\theta'_n = f(\theta_n) = \theta_n - \alpha \nabla \mathcal{L}^c_{\mathcal{P}train}(\theta_n)$$

이후엔 업데이트 된 θ'_n와 validation set 인 $\mathcal{D}_v^{train}$를 이용하여 loss 를 계산한 후 n번째 step 의 초기 가중치였던 θ_n를 업데이트한다.

$$\theta_{n+1} = \theta_n - \eta \nabla \mathcal{L}^v_{\mathcal{P}train}(f(\theta_n))$$

이러한 방식으로 최적의 가중치를 얻기 위해 수렴할 때까지 반복을 하게 된다.

C. Final Person-specific Adaptation

이전 과정까지의 결과로 최적의 모델인 $\mathcal{M}_{\theta^*}$가 나오게 되었으므로, 이제는 각 사람마다 그 사람에게 맞는 최적의 모델을 만드는 마지막 과정이 남아있다. $\mathcal{S}^{test}$ 에 포함된 $\mathcal{P}^{test}$ 데이터를 이용하여 $\mathcal{M}_{\theta^*}$를 fine-tuning 을 하게 되면 $\mathcal{P}^{test}$ 에 맞는 모델이 만들어지게 된다.

$$\theta_{\mathcal{P}^{test}} = \theta^* - \eta\nabla\mathcal{L}^{c}_{\mathcal{P}^{test}}(f(\theta_n))$$

3.2 Self-Authentication

감독관이 모든 시험자의 얼굴을 체크해야 하는 번거로움을 줄이기 위한 자동 얼굴 인식 기능이다. Self-Authentication 프로세스는 그림 에서 보이는 바와 같이 ① 사용자에게 필요한 정보를 받는 Pre-Information Registration 단계와 ② 시험을 치기 전 시험 대상자가 맞는지 판단하는 Authentication 단계로 나뉜다.

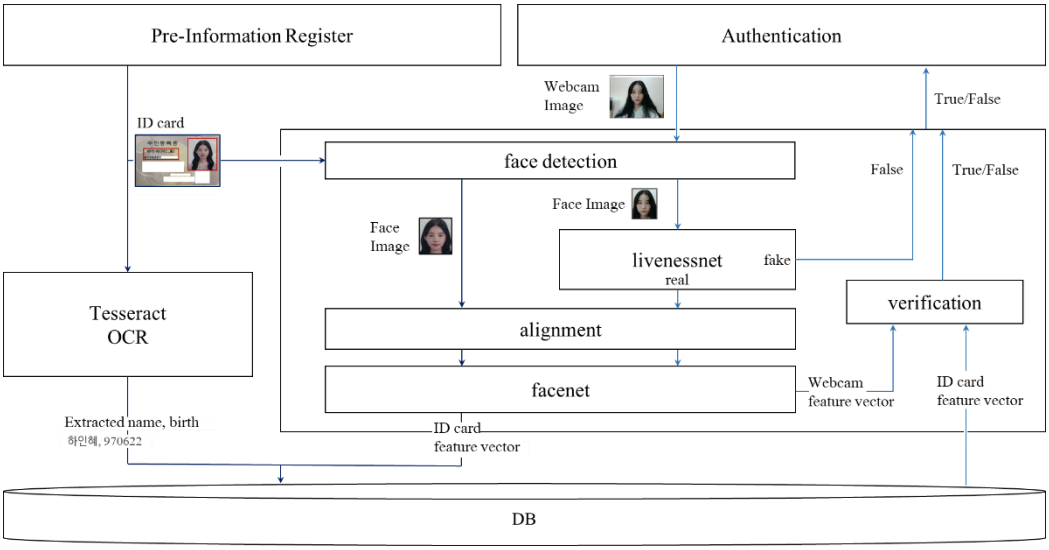


Figure 11. Authentication architecture

3.2.1 Pre-Information Registration -

A. Introduction

Pre-information registration 은 회원가입 시에 진행되며, 시험 치기 전 시험 대상자가 맞는지 확인하기 위한 사전 작업이라고 볼 수 있다. 이 과정에서 사용자의 이름, 생년월일, face feature vector 를 입력 받은 신분증에서 추출한 뒤 mongo DB 에 적재한다.

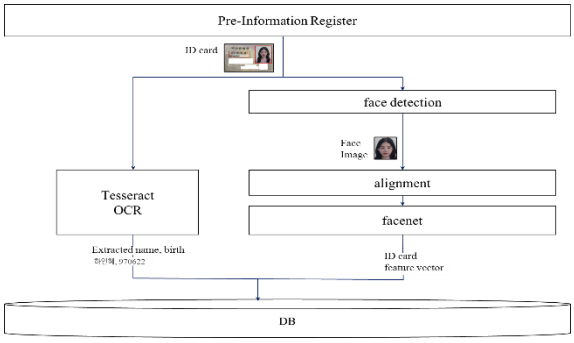


Figure 12. Pre-Information Process

B. Process

i. Extract name and birth

신분증 사진을 입력 받게 되면 Tesseract OCR을 사용하여 사진에서 이름, 생년월일을 추출한다. 사용자가 입력한 신분증은 개개인이 다양한 방법으로 개인정보를 마스킹하여 찍었기 때문에 이미지에 다양한 노이즈가 존재한다. 또한 Tesseract OCR 자체적으로도 전처리를 거치지 않으면 인식률이 낮아 신분증 사진에서 정확한 정보를 추출하기 위해서는 이미지 전처리가 필수적이다.

따라서 openCV를 사용하여 다양한 방법으로 이미지 전처리를 실험해 본 결과, Gray Scale을 적용한 뒤 Gaussian Blur 커널과 Median Blur 커널을 사용하였을 때 높은 인식률을 보였고 여러가지 버전으로 마스킹 된 신분증의 이미지에서도 가장 로버스트 했기 때문에 두 커널을 사용하여 전처리를 수행했다.

또한 신분증은 규격이 정해져 있는 점을 활용하여 이름과 생년월일의 위치에서만 문자를 추출하여 인식률을 개선하였다.

ii. Extract face feature vector

신분증의 인물 사진 부분을 미리 학습된 network에 넣어서 embedding 값을 구하는 과정이다. 이때 FaceNet 모델이 사용된다. face feature vector는 Authentication 단계에서 시험 대상자가 맞는 지 확인하는 verification 과정에서 사용되기 때문에 feature 정보가 사전에 구축이 되어 있어야한다. 따라 회원가입 정보들과 함께 DB에 저장된다.

(a) FaceNet

FaceNet 모델은 동일 인물일수록 임베딩 공간에서 이미지간 거리가 가깝고, 다른 사람일수록 이미지간 거리가 먼 것을 학습하여 동일인물인지 아닌지 판단한다.

FaceNet에서 사용하는 metric learning은 유사한 feature끼리는 가까운 거리로, 유사하지 않은 feature끼리 더 먼 거리로 판단하게 하는 어떤 metric을 학습한다. Metric learning이 성공적으로 이루어지면, 거리를 이용한 간단한 학습 모델을 이용하여 효율적으로 인식을 수행할 수 있다.

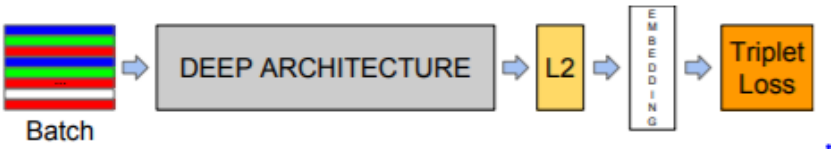


Figure 13. Model structure

① Loss Function

- *Triplet Loss*

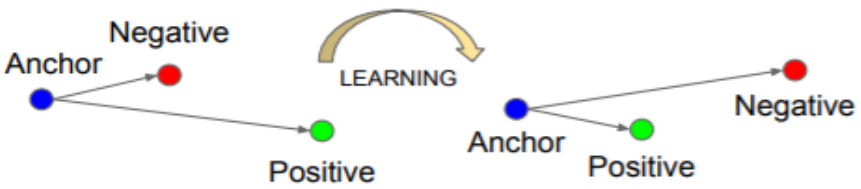


Figure 14. Triplet Loss

Triplet loss는 같은 사진은 가까운 거리에, 다른 사진 먼 거리에 있도록 유사도를 벡터 사이의 거리와 같아지게 하려는 목적으로 metric learning에서 사용되는 loss이다. Triplet loss는 Anchor와 positive(같은 사람), Negative(다른 사람) 3개의 sample에 대해 같은 이미지는 가깝게 다른 이미지는 멀게 loss 계산을 한다. Loss function은 아래와 같다.

$$\mathcal{L}_T = \sum_{i=0}^{N} \left[\left\| f\left(x_i^{anchor}\right) - f\left(x_i^{pos}\right) \right\|_2^2 - \left\| f\left(x_i^{anchor}\right) - f\left(x_i^{nag}\right) \right\|_2^2 + \alpha \right]$$

- *Triplet Selection*

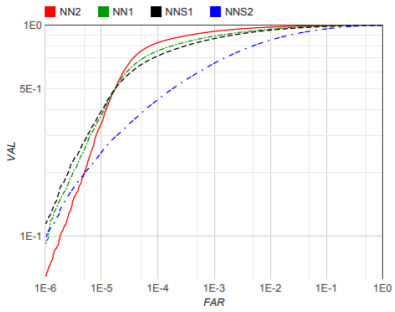
$$\text{argmax}_{x_i^p} \left\| f\left(x_i^{anchor}\right) - f\left(x_i^{pos}\right) \right\|_2^2 : \textit{Hard Positive}$$

$$\textit{argmax}_{x_i^p} \left\| f\left(x_i^{anchor}\right) - f\left(x_i^{nag}\right) \right\|_2^2 : \textit{Hard Nagative}$$

Hard Positive(같은 사람이지만 다르게 보이는 사람)과 Hard Negative(다른 사람이지만 닮은 사람)이 있는 경우 학습이 제대로 되지 않는다. 이를 방지하기 위해서 모든 anchor-positive쌍은 학습에 사용하고 hard negative는 아래 식을 만족하는 x중에 선택하여 학습한다.

$$\left\| f\left(x_i^{anchor}\right) - f\left(x_i^{pos}\right) \right\|_2^2 < \left\| f\left(x_i^{anchor}\right) - f\left(x_i^{nag}\right) \right\|_2^2$$

② *network*



architecture	VAL
NN1 (Zeiler&Fergus 220×220)	87.9% ± 1.9
NN2 (Inception 224×224)	89.4% ± 1.6
NN3 (Inception 160×160)	88.3% ± 1.7
NN4 (Inception 96×96)	82.0% ± 2.3
NNS1 (mini Inception 165×165)	82.4% ± 2.4
NNS2 (tiny Inception 140×116)	51.9% ± 2.9

Figure 15 . Network Architectures

그림 15는 FaceNet에서 여러 network들의 성능을 비교한 그래프이다.

본 프로젝트에서는 NN3를 사용하였으며, 따로 학습을 시키지 않고 pre-train되어 있는 가중치를 사용하였다.

3.2.2 Authentication –

A. Introduction

Authentication 단계는 시험치는 전 사전 점검시에 수행되며 현재 시험자가 시험 응시대상자 본인이 맞는 지 확인하는 단계이다. Authentication은 Anti-spoofing 기능과 verification 기능으로 구성되는

데 Anti-spoofing에서 합격해야 verification을 진행할 수 있다.

B. *Anti-Spoofing*

사람이 직접 신원을 확인하지 않고 프로그램으로 본인 확인을 할 때 생기는 문제점이 있는데 바로 spoofing attack이다. spoofing attack이란 출력된 사진이나 모바일 기기 내 촬영된 사진을 이용하여 허위로 얼굴 인증을 시도하는 행위이다. 따라 Anti-spoofing로 이러한 시도를 차단하도록 했다.

i. *Process*

Anti-spoofing의 프로세스는 ① *face detection*, ② *liveness model*, ③ *real/fake* 로 구성되어 있다.

(a) *Face detection*

먼저 webcam 이미지에서 얼굴 부분을 찾아 그 부분만 크롭하는 과정인 Face detection을 수행한다. 자주 사용되는 Face detector로는 OpenCV의 Haar Cascade, SSD(Single Shot-Multibox Detctor)과 Dlib의 HoG(Histogram of Oriented Gradients), MMOD(Max-Margin Object Detection)이 있다.

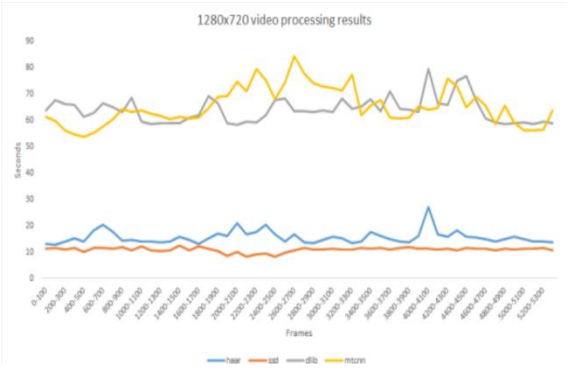


Figure 16. Face detectors performance

그림 13을 보면 알 수 있듯, SSD가 평균 9.20 frame을 처리할 수 있어 가장 빠르다. 따라서 본 프로젝트에서는 OpenCV의 DNN으로 SSD기반 face detector를 사용했다.

(b) *liveness model*

얼굴 부분만 크롭된 이미지를 실제 사람인지 아니면 spoofing을 시도한 것인 지를 판단하려면 미리 학습된 liveness model이 필요하다.

① *Dataset*

동영상에서의 anti-spoofing 데이터는 구하기가 힘들다. 따라 파이썬 스크립트를 사용하여 약 200개가량의 positive 영상들을 다운 받고 그 후 영상에서 OpenCV의 DNN으로 SSD기반 face detector로 얼굴만 추출해 데이터를 저장했다. Negative 데이터는 positive 영상을 여러 기기 (iphone 7, iphone X, iphone11 pro, ipad pro 3rd generation, samsung S10)에서 재생하고 이를 촬영하여 negative 데이터로 사용하였다.

한편 얼굴을 추출할 때 동영상은 연속된 프레임이기 때문에 동일한 얼굴만 추출되어 overfitting이 될 수 있는 가능성이 있다. 따라 연속된 프레임만을 얻는 것을 피하기 위해서 16 frame씩 건너뛰어 데이터를 추출하였다.

② *Network*

Liveness model는 간단한 CNN을 사용하며 VGGNet과 비슷하다. Network는 2개의 conv 블록과 1개의 fully connected layer를 가지며 깊이가 매우 얕다. 얕은 깊이의 network를 사용함으로써 spoofing 감지를 실시간으로 빠르게 실행할 수 있다.

③ *Evaluation Matric*

Anti-spoofing의 성능 평가에 APCER(Attack Presentation Classification Error Rate), NPCER(Normal Presentation Classification Error Rate), ACER(Average Classification Error Rate)는 평가 지표로 사용된다.

Attack Presentation Classification Error Rate (APCER):

$$APCER = FP / (TN + FP)$$

Normal Presentation Classification Error Rate (NPCER):

$$NPCER = FN/(FN + TP)$$

Average Classification Error Rate (ACER):

$$ACER = (APCER + NPCER) / 2$$

C. Verification

FaceNet Model을 사용하여 verification을 수행한다. FaceNet 논문에서는 전통적인 Verification 파이프라인(① detection ② alignment ③ representation ④ verification)에서 detection과 alignment 과정 없이 verification을 수행한다.

하지만 본 프로젝트에서는 모델의 성능을 높이기 위해 추가적으로 detection 과 alignment를 해주었다. 따라 Verification 파이프라인은 ① detection ② alignment ③ representation ④ verification으로 구성된다. Detection 과정은 앞 anti-spoofing과 동일하다.

(a) Detection

Detection은 앞에서 설명한 anti-spoofing과 동일하다. Webcam의 이미지에서 얼굴 부분만 추출한다.

(b) Alignment

Alignment는 간단하게 말하면 눈과 코 등 얼굴의 특징을 나타내는 점을 찾는 과정이다. OpenCV의 haarcascade eye detector를 사용하여 두 눈을 찾아 사이 각을 사용하여 얼굴을 정렬하는 방식을 사용했다.

(c) Representation

Pre-information Registration에서와 같이 FaceNet model의 network를 사용하여 aligned 된 얼굴 이미지에서 feature vector를 추출한다.

(d) verification

DB에 저장되어 있던 사용자의 feature vector를 불러와 representation 단계에서 구한 webcam에서의 feature vector와의 cosine similarity를 구한다. Cosine similarity가 threshold인 0.45보다 크게 되면 False를 반환하고 작으면 True를 반환한다.

D. Authentication architecture

Authentication의 파이프라인은 Verification의 파이프라인(① detection ② alignment ③ representation ④ verification)에서 anti-spoofing이 추가된 ① detection ② anti-spoofing ③ alignment ④ representation ⑤ verification 이 된다. 최종적으로 완성된 Authentication 모델의 구조는 그림과 같다.

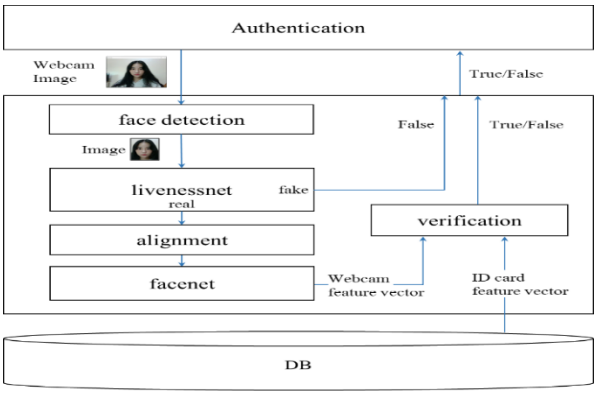


Figure 17. Authentication architecture

4 TECHNOLOGY STACK

4.1 Frontend

4.1.1 TypeScript

A. Characteristics

타입스크립트는 MS 에서 개발하고 관리하는 오픈소스 프로그래밍 언어로 어떤 브라우저나 호스트, 운영체제에서도 동작한다. 기존 JavaScript(ES6)의 모듈, 네임스페이스를 모두 포함한 데다가 JavaScript 의 특징인 동적 타입 언어의 단점을 보안한 언어라고 할 수 있다. 이름 그대로, TypeScript 는 타입을 지정하여 타입 안정성을 보장할 수 있다. 하지만 동적 타입 언어의 장점인 런타임속도는 JavaScript 에 비해 성능이 떨어진다.

B. Reason for using TypeScript

JavaScript 의 가장 큰 문제점인 동적 타입을 지원하여 개발자의 의도와 다른 형태의 오류를 잡아내지 못한다. 따라서 소규모 프로젝트에는 JavaScript 를 사용하여도 대조하여 오류를 찾을 수 있지만, TypeScript 는 정적 타입을 지정해야 하기 때문에 컴파일 단계에서 오류를 포착할 수 있게 해주고, 코드 작성에는 번거로움이 늘지만 개발자의 의도를 명확하게 코드로 기술할 수 있고, 대규모 프로젝트에 어울린다.

4.1.2 React

A. Characteristic

React 는 페이스북이 만든 UI 개발 라이브러리로 간단한 HTML, CSS 만으로 이루어진 웹페이지를 넘어, SPA(Single page application)개념을 도입하여 이벤트가 발생하였을 때 페이지 리로딩이 일어나지 않고 변경된 데이터만 렌더링 되고 이는 UI/UX 면에서 큰 장점으로 작용한다.

B. Reason for using React

위에서 언급한 장점(SPA)을 가진 Frontend framework(library)는 많다.(Angular, Vue 등) 그 중에서도 React 를 사용하는 이유는 가장 대중적이고 활용성이 높기 때문이다. 20 년 기준으로 ReactJS 는 다른 Framework 에 비해 가장 많은 개발자가 사용하는 라이브러리이고 가장 대중적인 라이브러리이기 때문에 가장 큰 커뮤니티가 형성되어 있고, 참고할 정보가 많다. 추가로, 온라인 시험 테스트 플랫폼은 모바일 앱으로 까지 확장할 필요성이 생길 수 있다. React 는 React Native 를 통해 상대적으로 간편하게 모바일 앱을 생성할 수 있다는 장점이 있다.

4.1.3 Redux

A. Characteristic

Redux 는 가장 사용률이 높은 상태관리 라이브러리이다. 상태관리 라이브러리의 역할은 A component 가 B component 에, B component 가 C component 에, 이런 방식으로 G 컴포넌트 까지 이어진 루트가 있다면 A component 가 G component 로 접근하려면 A -> B -> C -> D -> E -> F -> G 형태로 접근해야 한다. 이때 상태관리 라이브러리를 사용하면 A -> Store -> G 형태로 접근할 수 있게 해준다. Redux 는 React 의 이러한 고질적인 단점을 해결해주기 때문에 React 에서 자주 사용되긴 하지만 React 내부에 있는 라이브러리는 아니다.

B. Reason for using Redux

React 프로젝트의 가장 큰 단점은 데이터 상태관리에 있다. React 는 상태관리를 State, Props 두 가지의 형태로 이동하기 때문에 대규모 프로젝트로 진행될수록 데이터 이동이 복잡해지고 번거로워진다. Redux 는 이런 단점을 해결해주기 때문에 React 와 자주 사용된다.

4.2 Backend

4.2.1 Django

A. Characteristic

Django 는 Python 기반의 무료 오픈소스 웹 어플리케이션 프레임워크로 Flask 와 달리 일정한 틀을 제공하여 사용자 인증, 사용자 관리, 데이터베이스 등을 쉽고 간편하게 제공한다. 또한 이러한 특징 덕분에 익히기 쉬워 개발 속도가 빠르고, 일정한 틀을 제공하기 때문에 코드 해석, 코드 완성도면에서 높게 유지할 수 있으나 개발자들이 많이 사용하지 않는 코드에 대해서는 구현이 어렵고, 성능이 다른 웹 어플리케이션에 비해 좋지 않다는 단점이 있다.

B. Reason for using Django

ML/DL 의 일반적인 언어인 파이썬을 사용하여 다른 언어를 이용한 프레임워크(Node.js, Spring 등)에 비해 ML/DL 모델과의 호환성이 좋다.

4.2.2 Django Channels, Redis

A. Characteristic

Django Channels 는 Websocket 을 사용하기 위해 사용하는데, 사용자 브라우저와 서버사이의 동적인 양방향 연결채널을 구성한다. Websocket API 를 통해 서버로 메시지를 보내고, 요청 없이 응답을 받아오는 것이 가능한다. 이를 사용하기 위해 ASGI 라는 Python 사양을 기반으로 한다.

B. Reason for using Django

Websocket 을 사용해야하는 이유는 Eye-Tracking Model 을 사용할 때, 실시간 시험 플랫폼을 이용할 때 등 서로 실시간으로 비디오 프레임을 보내야하기 때문이다. Websocket 을 사용하지 않고 HTTP 요청으로 보내게 되면 서버에서 동영상을 보내는 정보를 알 수 없기에, 사용자 브라우저와 서버를 1:1 로 연결한 뒤 영상을 보낸다. 이는 서버에 이미지를 보내는 것이 아닌 동영상, 웹 캠 영상 등을 보낼 때 일반적으로 사용하는 방식이다.

4.2.3 OpenCV (Open Source Computer Vision)

A. Characteristic

OpenCV 는 컴퓨터 비전을 목적으로 한 프로그래밍 라이브러리이다. 이미지 프로세싱을 위한 라이브러리로, 기본적으로는 C/C++ 코드이나, Python 라이브러리로도 지원한다. Python 에서 수학 연산에 많이 사용하는 Numpy 라이브러리와 호환이 잘 되며, 컴퓨터 그래픽스, 이미지 프로세싱, 비전 처리 등에 필요한 행렬연산을 손쉽게 할 수 있다. 뿐만 아니라 딥러닝 FrameWork 인 TensorFlow, PyTorch 를 지원한다.

B. Reason for using OpenCV

본 프로젝트에서는 사진에서 글자를 인식하기 위한 이미지 전처리, face detection & alignment, 실시간으로 사용자의 웹 캠 이미지를 받아 왜곡 제거를 위한 Camera Calibration, Head Pose Estimation 등을 위하여 사용되었다.

4.2.4 Tesseract OCR (Optical Character Recognition)

A. Characteristic

Tesseract 는 1984~1994 년에 HP 연구소에서 개발된 오픈 소스 OCR 엔진이다. 이미지로부터 텍스트를 인식하고, 추출하는 소프트웨어를 일반적으로 OCR 이라고 한다. Tesseract 는 현재까지 LSTM 과 같은 딥러닝 방식을 통해 텍스트 인식률을 지속적으로 개선하고 있다. 입력 이미지가 이미지에 맞게 사전 처리되지 않은 경우 인식률이 매우 떨어진다. 하지만 충분한 전처리를 거친다면 다른 OCR 도구들 보다 높은 인식률을 보인다.

B. Reason for using Tesseract OCR

본 프로젝트에서는 신분증 사진에서 생년월일과 이름을 추출하기 Tesseract OCR 을 사용했다.

4.2.5 PyTorch

A. Characteristic

본 프로젝트는 파이썬 프로그래밍 언어를 이용해 만들었는데, 파이썬 프로그래밍 언어에서 딥러닝 모델을 사용하기 위한 프레임워크로 크게 TensorFlow, PyTorch 가 존재한다. 그 중 , PyTorch 는 페이스북의 AI 연구 팀이 만든 딥러닝 프레임워크로, Define by run 방식으로 코드를 깔끔하고 직관적으로 작성할 수 있고 디버깅이 상대적으로 쉽고 코드 커스텀도 쉽다는 장점이 있다. 또한 Numpy 를 대체하면서도 GPU 연산이 가능하며 유연하고 빠르기 때문에 최근 딥러닝 관련 논문에서 많이 사용이 되고 있다.

B. Reason for using PyTorch

본 프로젝트에서 Eye-Tracking 관련 모델이 PyTorch 로 작성이 되어있어 사용하게 되었다.

4.2.6 *TensorFlow*

A. Characteristic

TensorFlow 는 파이썬 프로그래밍 언어에서 딥러닝 모델을 사용하기 위한 프레임워크 중 하나로 대중적으로 쓰인다. 구글 브레인팀이 개발하였고 장점으로는 파라미터의 변화 양상이나 DNN 에 대한 구조도를 텐서 보드로 그래프로 확인이 가능하기 때문에 Tensor 들과의 연결관계, Tensor 의 Flowing Status 를 쉽게 파악할 수 있다. 또한 커뮤니티가 형성되어 있어 여러 레퍼런스들이 존재한다.

B. Reason for using PyTorch

Authentication 관련 모델들의 레퍼런스가 TensorFlow 를 프레임워크로 사용하는 경우가 많아 사용하게 되었다.

4.3 *MongoDB*

A. Characteristic

MongoDB는 문서지향 저장소를 제공하는 NoSQL 데이터베이스 시스템으로, 현존하는 NoSQL 데이터베이스 중 인지도 1위를 유지하고 있다. NoSQL이란 Not Only SQL의 줄임 말로 SQL과 달리 단순한 키와 값의 쌍으로 이루어져 있다. 인덱스와 데이터는 분리되어 별도로 운영되며 고정된 스키마도 없다. 이는 데이터를 유연하게 처리하는 것에 능숙하다고 볼 수 있다.

B. Reason for using MongoDB

이 웹 어플리케이션은 아이디, 비밀번호, 신분증을 받고 유저를 생성시키고, 신분증에 있는 생년월일, 이름 사진 등을 그 후 추출하여 다시 유저를 변경시킨다. 사진도 임베딩하여 유저의 정보를 변경시킨다. 이러한 환경에서는 엄격한 SQL보다는 여러가지 유연한 정보의 상태를 허용하는 NoSQL형태의 데이터베이스가 더욱 적절할 것이라 판단하여 MongoDB를 사용하였다.

5 PROJECT ARCHITECTURE

5.1 *Online Test Platform*

5.1.1 *Register/Login*

A. Introduction

회원가입 단계에서는 사용자의 학번, 비밀번호, 비밀번호 확인, 신분증, 신분을 선택할 수 있다. 특이한 점은 신분증에서는 본인의 신분증을 이름과 생년월일을 제외한 부분은 지우고 업로드 해야 정보의 유출위험이 적다. 신분 카테고리에서는 학생 혹은 관리자로 선택할 수 있는데, 학생으로 선택하면 테스트 페이지에서 Authentication,

Figure 18. Register/Login Page

Calibrate, Collect, Screensharing 4가지 과정을 거치고 테스트 화면으로 가지만 관리자로 선택하면 학생을 관제할 수 있는 페이지로 바로 넘어간다. 빈칸을 채워넣고 제출 버튼을 누르면 로딩 화면이 나타나고, 로딩화면 이후에 OCR에서 인식한 이름과 생년월일을 확인하는 페이지로 이동한다. 이후 확인과정을 거치면 자동으로 로그인을 하게 되고 홈 화면으로 이동한다.

로그인 단계에서는 회원가입 된 계정의 학번, 설정한 비밀번호를 입력하면 확인 과정을 거친 후 홈 화면으로 간다.

B. *Frontend (UI/UX, CSS)*

i. *Register*

회원가입 단계의 HTML은 form 형태로 이루어져 있다. Form 안에 div를 리스트로 나타낸 형태로 display는 grid를 이용하여 세로로 정렬하였다. 제출을 클릭하면 onSubmit을 통하여 비밀번호의 조건들을 확인하고 모든 것들이 일치하게 되면, 백엔드로 정보를 보낸다. 정보를 보낼 때, 신분증 이미지를 같이 보내야 하므로 FormData를 사용하여 FormData에 이미지와 학번, 비밀번호, 신분 등을 넣고 백엔드로 보낸다.

로그인 단계도 마찬가지이지만, 회원가입과 달리 이미지를 보낼 필요가 없기 때문에 FormData를 사용하지 않고 JSON파일로 보낸다.

ii. *Redirect*

회원가입 단계에서 제출을 누르면 제출하고 다시 회원가입 페이지로 온다. 일반적으로 회원가입 단계에서 제출을 누르면 로그인 페이지로 가거나 로그인까지 자동으로 해서 홈으로 가는 경우가 일반적이기 때문에 이를 작업할 필요가 있었다. 따라서 회원가입 요청을 보내고 성공하면, 로그인 함수를 실행하고 로그인이 성공하면, 회원가입 시 OCR로 인식한 생년월일과 이름을 확인하는 페이지로 Redirect한다.

로그인 단계에서는 로그인이 성공하면 홈 화면으로 Redirect한다.

iii. *Loading*

OCR과 신분증 사진을 데이터베이스에 임베딩 하는데, 3~4초 정도 시간이 소요된다. 이를 아무런 조치를 하지 않으면 사용자는 페이지가 안 넘어가는 거라 착각하게 되므로 로딩 화면을 넣을 필요가 있었다. 이는 간단하게 react-loader-spinner라는 라이브러리를 설치하여 해결하였다.

iv. *Focus*

회원가입 단계에서 학번, 비밀번호 등을 입력할 때 클릭하면 학번 글자가 올라가고 줄이 강조가 되는 애니메이션 효과가 있다. 그 애니메이션이 발생하는 조건은 클릭을 하였을 때(다시 클릭하면 전 상태로 돌아감), 클릭하지 않더라도 Input에 글자가 입력되었을 때 발생한다. 애니메이션 효과는 CSS의 transition을 통해 적용하였다. 변수를 하나 생성하고 False로 만든 다음, 위 조건을 만족하면 변수를 True로 변경한다. 이후 변수가 True이면 HTML태그에 새로운 class를 적용하고 변경사항을 transition으로 속도를 정하면 자연스럽게 효과가 나타난다.

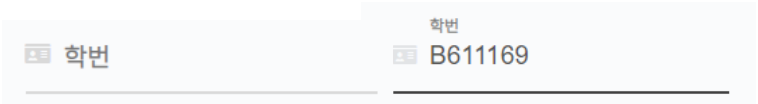


Figure 19. Focus Animation

C. *Backend*

i. *Network Communication*

(a) *HTTP vs WebSocket*

HTTP란 HTML 파일을 전송하는 프로토콜이라는 의미를 가진다. 초기에는 HTML파일을 전송하려는 목적이었으나 현재는 JSON, Image파일 등을 전송하는데 사용된다. HTTP통신은 사용자에서 서버로 요청을 보내고 서버가 응답하는 방식으로 이루어진다. 이는 사용자만이 요청을 보낼 수 있는 단방향 통신이다. 간편한 JSON파일 혹은 이미지를 주고받는 소통에는 HTTP 통신이 유지보수 면에서 이득이 있다.

HTTP와 대비되는 통신은 WebSocket 혹은 Socket 통신이다. 소켓이란 두 프로그램이 서로 데이터를 주고 받을 수 있도록 양쪽에 모두 생성되는 통신 단자이다. 또한 사용자와 서버가 계속 연결을 유지하고 있기 때문에 Streaming 중계나 실시간 채팅과 같이 즉각적으로 정보를 주고받아야 하는 경우 유리하다.

	HTTP	WebSocket
장점	실시간 연결이 아닌 필요한 경우에 요청을 보내는 상황에 비용 및 유지보수에 유리	실시간 연결일 경우 서버가 사용자를 기억하고 있고, 양방향 통신이기 때문에 비용에서 유리
단점	실시간 연결일 경우 서버가 사용자를 기억 못하고, 응답을 계속 보내야 하기 때문에 비용에서 불리	사용자와 서버가 계속 연결을 유지하기 때문에 비용 및 유지보수에 불리

(b) *Register/Login HTTP communication*

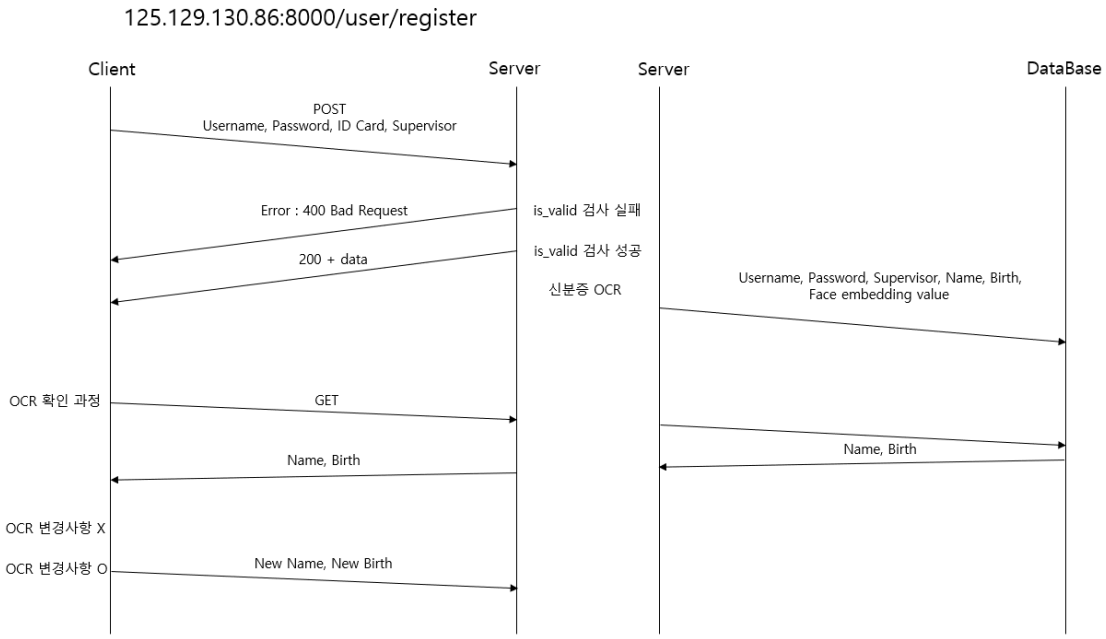


Figure 20.Register HTTP Communication

Register, Login의 통신방법은 HTTP통신 방법을 이용한다. 학번, 비밀번호 등 POST형태로 보내게 되면 서버에서 적절한 값이 왔는 지 확인하는 과정을 거친다. 적절한 값이 왔다면, Authentication Model에서 OCR하는 과정을 거치고 이름, 생년월일, Face Embedding 값을 추

출한 뒤 데이터베이스에 저장한다. 이후 사용자는 OCR 값을 확인하기 위해서 GET요청을 보내고, 이름, 생년월일 값이 잘못되면 다시 OPTION 요청으로 새로운 이름과 새로운 생년월일을 업데이트한다.

ii. Models

생성된 객체는 User 모델에 넣게 된다. User model은 Django에서 기본적으로 제공하는 Abstract User 항목과(username, password 등), name, birth, image, face embedding, supervisor항목을 포함한다. 회원가입 단계에서 학번, 비밀번호, 신분증 사진, 신분이 username, password, face embedding, supervisor에 대응한다.

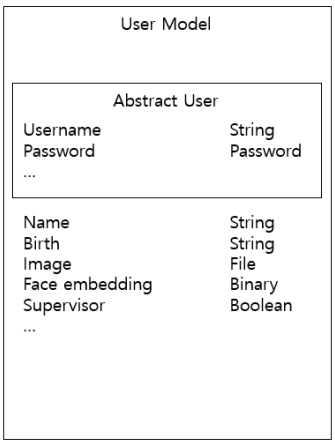


Figure 21. User Model

iii. Serializers

REST API를 사용하거나 요청, 응답을 보낼 때 데이터는 필히 문자열로 표현되어야 한다. 각 언어에서 모두 지원하는 직렬화 포맷인 JSON을 이용하는 것이 일반적인데 파이썬 객체를 JSON으로 변경하는 과정이라고 보면 된다. Django에서는 Meta 클래스를 통해 자동으로 JSON으로 변경해주는 기능을 포함하기 때문에 Meta 클래스에 원하는 항목만 기입하면 된다.

iv. Views

Views에서는 실제 사용자에게서 받은 정보를 처리하는 곳이다. 따라서 User 세션에서는 Register, Login, User, Logout 4가지를 처리하고, 이는 다음과 같다.

(a) RegisterView(/user/register)

Register에서는 JSON파일이 조건에 다 맞는 지 확인하는 valid과정을 거치고 오류가 없다면 데이터베이스에 저장한다. 이후 Response를 사용자에게 보낸다.

(b) LoginView(/user/login)

Login에서는 받아온 username과 password를 데이터베이스에 동일한 username이 있는지 확인하는 과정을 거치고, 패스워드가 맞는 지도 확인한다. 모두 만족한다면, HS256알고리즘을 이용한 JWT토큰을 발급하여 쿠키를 포함하여 Response한다. 이 쿠키를 통해 사용자는 홈 페이지 접근과 사용자 정보 등 제어할 수 있게 된다.

(c) UserView(/user/user)

User에서는 로그인 한 유저의 정보를 Response합니다. 먼저 요청한 쿠키를 확인하여 쿠키에 있는 JWT를 디코딩합니다. 디코딩 한 정보를 통해 데이터베이스에서 User의 정보를 받고, Response로 정보를 보내준다.

(d) LogoutView(/user/logout)

Logout에서는 받은 쿠키를 지우고 Response를 한다.

5.1.2 Navigation Bar

Figure 22. Navigation Bar

A. *Introduction*

웹페이지에 흔히 있는 것으로 사용자가 원하는 페이지로 이동할 수 있게 해주는 역할을 한다. 상단에 위치하며 네비게이션 바의 좌측에는 메인 아이콘, 우측에는 Test, Result, Settings, Logout 페이지로 이동할 수 있게 한다.

B. *Frontend*

i. *Current Location*

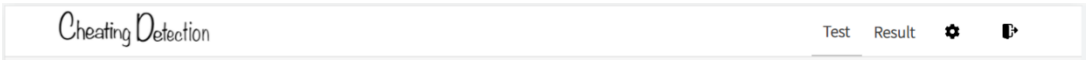


Figure 23. Navigation Bar with Border Line

현재 위치를 알아볼 수 있는 밑줄기능이다. 이는 각각의 아이템에 변수를 하나씩 지정한 뒤, 현재 URL을 비교하면서 해당 변수에만 True값을 주어서 밑줄기능을 추가하였다.

ii. *Keeping Space*

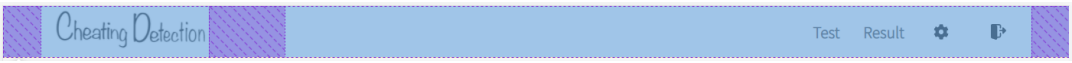


Figure 24. Navigation Bar CSS

CSS에서 가장 주목해야 할 점은 호환성이다. 사용자들마다 이용하는 모니터, 브라우저의 크기가 다르기 때문에 다양한 브라우저에 대한 호환은 필수적이다. 따라서 CSS 작업할 시에는 픽셀로 설정하는 것을 경계하고 브라우저 크기에 따른 상대적인 크기를 설정하는 것이 중요하다. 따라서 브라우저를 기준으로 하는 크기인 vm, vh 혹은 %단위로 설정하고, display를 이용하여 배치하는 것이 요령이다. 이 네비게이션 바는 display를 flex로 설정하고, space between으로 하여 브라우저의 크기가 변경되어도 양 옆의 일정한 간격을 유지하도록 하였다.

5.1.3 *Home*

A. *Introduction*

회원가입/로그인 후 나타나는 첫번째 화면으로 프로젝트를 설명하고 프로젝트의 진행방식을 전체적으로 설명하는 페이지이다.

B. *Frontend*

i. *3x3 Block*

프로젝트의 목표를 아이콘과 키워드로 간단하게 표현한 것이다. 이는 Grid display를 이용하여 3 column, 3 row로 생성한 뒤 각각에 적절한 아이콘과 키워드를 넣어서 생성하였다. 아이콘은 Font Awesome에 있는 무료 아이콘을 이용하였다.

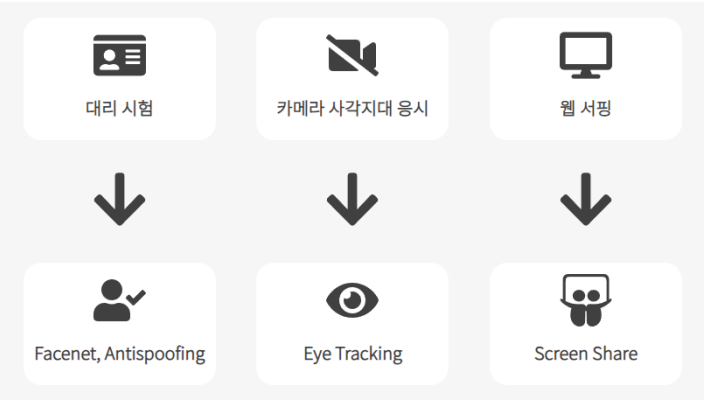


Figure 25. 3x3 Block

ii. Guide

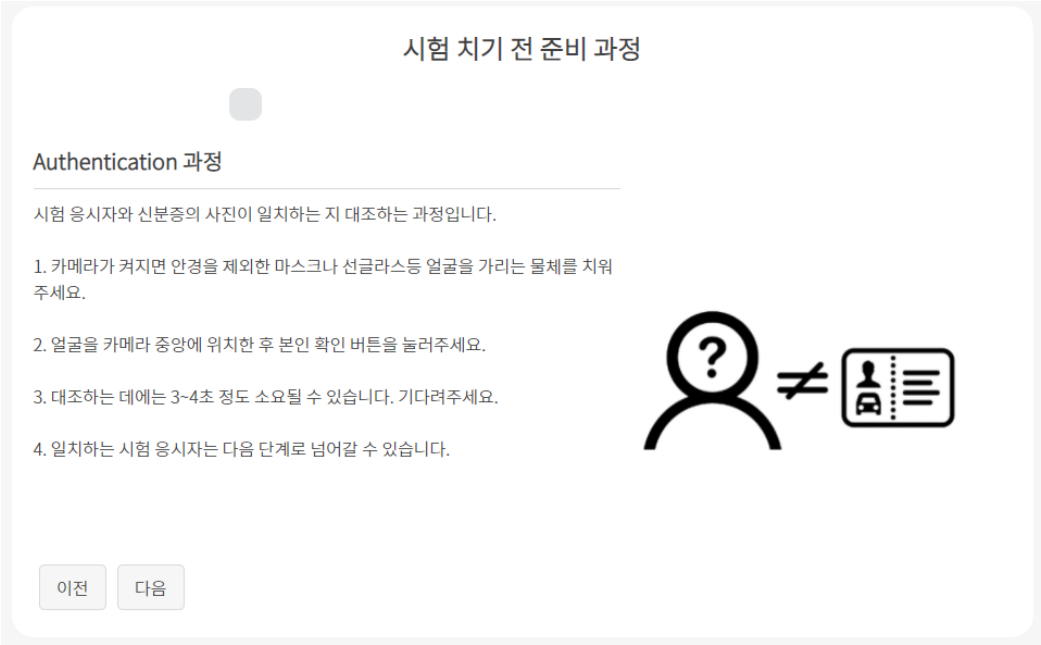


Figure 26. Guide in Main Page

시험 치기 전 준비 과정을 어떻게 해야 하는지 안내해주는 과정이다. 상단에는 회색 블록이 다음을 누를 때마다 크기가 커지는데 회원가입 기입란과 마찬가지로 transition을 이용하여 증가하는 애니메이션을 표현한다.

5.1.4 Settings

A. Introduction

Settings에서는 사용자의 프로필, 사용자의 프로필 변경, 비밀번호 변경을 할 수 있는 페이지이다. 프로필에서는 사용자의 학번, 이름, 생년월일, 가입 일을 확인할 수 있고, 프로필 변경은 이름과 생년월일 변경, 비밀번호 변경은 기존 비밀번호를 입력하면 새 비밀번호로 변경한다.

B. Frontend

i. Sidebar Navigation

사이드바 네비게이션은 좌측에 위치하고, Grid display를 통해 좌, 우로 화면을 분할하고 좌측에 네비게이션 바를 넣었다. 마우스를 위치하거나 자신이 위치한 곳에 진한 색깔이 나타난다. Hover를 이용하여 마우스 위치를 감지하였고, 현재 URL을 확인하여서 위치를 탐지하였다.

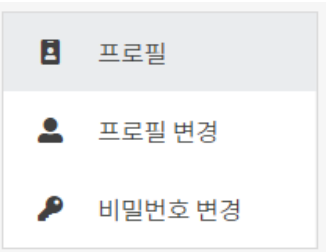


Figure 27. Sidebar Navigation

ii. Change View

프로필, 프로필 변경, 비밀번호 변경의 형식은 3가지 모두 같은데, 박스 형태가 아닌 제목 하단에 Border를 놓는 형식으로 디자인하였다. 이후 Grid display로 세션을 나누고 좌측은 우측 정렬, 우측은 좌측 정렬하였다.

Figure 28. My Profile in Settings Page

C. Backend

i. Views

회원가입/로그인과 동일한 User 세션을 이용하고 프로필 변경, 비밀번호 변경을 추가하였다.

(a) UpdateNameView

Django에서는 모델의 변경(update)를 제공한다. 따라서 사용자가 바꾸고 싶은 이름과 생년월일을 입력하면, 요청 할 때 쿠키에서 가져온 학번을 포함하여 요청을 보내면, 서버에서 학번을 통해 유저를 찾고, 이름과 생년월일을 업데이트한다.

(b) UpdateView

위의 UpdateNameView와 동일하지만 비밀번호 보안과정을 거쳐서 하는 점이 다르다. 비밀번호를 암호화하는 과정을 거쳐서 기존 비밀번호에 새로운 비밀번호를 덮어쓰기 한다.

5.1.5 Test

A. Introduction

i. Supervisor

Supervisor로 회원가입한 사람의 Test 과정에서는 시험치는 학생들을 관제할 수 있다. 좌측에는 학생의 화면공유 화면과 학생의 웹 캠 화면이 나타나고, 우측에는 학생 학번, 학생과의 채팅, 학생의 시선이탈 알람 로그가 나타난다.

ii. Student

Student로 회원가입한 사람의 Test 과정에서는 부정행위를 방지하기 위한 Pre-Check 과정을 4개 거치고 시험 치는 화면으로 넘어간다. Pre-Check 과정에는 Authentication, Calibrate, Collect, Screen Sharing과정이 있고 Authentication은 대리시험 방지, Calibrate, Collect는 카메라 시선 이탈 방지, Screen Sharing은 웹 서핑 방지 역할을 한다. Pre-Check 과정을 모두 거치면 실제 시험 치는 화면으로 넘어가는데, 우측에는 본인 웹 캠, 문항을 체크할 수 있는 답안지, 관리자와 체크할 수 있는 채팅창이 있고, 좌측에는 시험 문항이 있다.

B. Frontend

i. WebRTC(Websocket + P2P Connection)

(a) Why WebRTC?

WebRTC란 브라우저나 모바일 어플리케이션에서 Real Time Communication(RTC)를 편리하게 할 수 있는 API를 만들고자 하는 목적으로 시작된 프로젝트이다. 이는 브라우저에서 Real Time으로 하는 것들을 제공하는데, 대표적으로 채팅, 영상, 음성을 서로 주고받는 것에 있다. 따라서 학생들의 영상, 화면 영상, 채팅 등을 관리자에게 전달하고 서로 상호작용하기 위해서 WebRTC를 사용하는 것이 일반적이다.

(b) WebRTC's Representative Three APIs.

① MediaStream

사용자의 카메라와 마이크의 데이터 스트림에 접근한다. 우리의 어플리케이션이 사용자의 음성, 영상 데이터를 채집해올 때 자주 이용된다.

② RTCPeerConnection

암호화 및 대역폭 관리를 하는 기능을 가지고 있고, 오디오 또는 비디오 연결을 담당. 어플리케이션이 채집한 음성 및 영상 데이터를 서로 주고 받는 채널을 추상화한다.

③ RTCDataChannel

음성 및 영상 데이터가 아닌, json/text 데이터들을 주고받는 채널을 추상화한 API

(c) What WebRTC does.

WebRTC를 사용하게 된다면 다음의 주요한 4가지 작업을 수행하게 될 것이다.

No	*Stage*	*Description*
1	*Fetching*	상대 Peer 에게 보낼 사용자의 음성 및 영상 데이터를 수집
2	*Signaling*	상대 Peer의 정보를 탐색
3	*Connection*	발견한 Peer와 P2P Connection을 맺는다. Channel을 개방
4	*Communication*	개방한 채널을 통해 음성/영상/텍스트 데이터 주고받기

① Fetching

WebRTC API인 MediaStream, getUserMedia를 이용하여 사용자의 영상 및 음성 정보를 가져온다. 웹 캠의 경우 getUserMedia을 통해 가져오고, 화면 공유의 경우는 getDisplayMedia를 통해 가져올 수 있다.

② Signaling

Signaling 단계는 서로 다른 두 peer가 Communication 하기 위한 준비단계로 3가지 종류의 정보를 교환해야 한다. 첫번째는 Network 정보를 교환한다. ICE Framework를 통해 ip와 port를 찾는 단계를 거치고, 두번째로 Media Capability를 교환한다. sdp(Session Description Protocol) 형식을 따르는 blob인 offer와 answer을 주고받으며 교환한다. 마지막으로 Session Control Messages를 교환한다. Session의 초기화, 종료를 담당한다. 복잡한 내용이니 조금 더 자세히 설명하자.

- *Exchange Network Information (ICE Candidate)*

  상대 peer를 찾아 연결을 맺기 위해서는 네트워크 정보를 교환해야한다. 이 때 중간 매개자 역할로서 별도의 서버인 Signaling Server가 필요하다. 이 때 보통 Websocket을 사용하여 new offer, new answer등을 생성하는데, 이는 Backend단계에서 자세히 설명한다. 네트워크 정보를 교환하는 순서는 다음과 같다.

  Step 1: RTCPeerConnection Object를 새롭게 생성하고, 안에 내제된 함수인 onicecandidate 핸들러를 통해 현재 내 client의 Ice Candidate(Network 정보)가 확보되면 실행될 callback을 전달한다.

  Step 2: Ice Candidate(내 네트워크 정보)가 확보되면, 중간 매개자인 Signaling Server를 통해 상대 peer에게 자신의 네트워크를 전송한다. 마찬가지로 상대방도 자신의 네트워

크 정보를 전달한다.
Step 3: 상대 peer의 네트워크 정보가 도착하면 RTCPeerConnection 안에 내제된 함수인 addIceCandidate를 통해 상대 peer의 네트워크 정보를 등록한다. (이것도 쌍방으로 진행된다.)

- *Exchange Media Capability + Exchange Session Control Message*

서로가 서로의 네트워크 정보를 교환하고, addIceCandidate를 통해 서로의 네트워크 정보를 등록한 다음은 어떻게 해야 할까? A와 B가 통신을 한다고 가정한 뒤 그 이후의 순서는 다음과 같다.
Step 1: B가 RTCPeerConnection의 createOffer을 호출해 offer sdp를 생성한다. 여기에는 브라우저에서 사용 가능한 코덱이나 해상도에 대한 정보가 들어있다.
Step 2: B가 Offer SDP를 Signaling Server를 통해 전송한다.
Step 3: A는 Signaling Channel에서 Offer SDP를 받아, RTCPeerConnection의 setRemoteDescription을 수행한다.
Step 4: A의 RTCPeerConnection 객체는 상대는 상대 session에 대한 정보를 알고 있게 되었고, RTCPeerConnection의 createAnswer를 호출하여 Answer SDP를 생성하여 Signaling Channel을 통해 B에게 전달한다.
Step 5: B도 자신의 setRemoteDescription을 호출해 전달받은 Answer SDP를 등록한다.
Step 6: A, B각 측에서 성공적으로 수행되었다면, P2P 연결이 성공적으로 완료된 것이라 볼 수 있다.

③ *Connection*

Signaling을 통해 상대 peer의 정보가 잘 등록된 RTCPeerConnection을 얻었다면, 연결이 이루어진 것이다.

④ *Communication*

연결이 이루어졌다면, 서로의 데이터를 주고받게 되는데 크게 두 개의 종류의 데이터를 주고받는다. Video나 Audio 데이터 스트림 혹은 Serialized된 json/text데이터인데, 두 종류의 데이터 주고받는 방식이 각각 따로 있다.

- *Video/Audio Data Stream*

주는 입장: getUserMedia/getDisplayMedia를 통해 video/audio 스트림 source를 취득해 RTCPeerConnection을 생성할 당시에 addTrack을 해준다. Signaling을 통해 connection을 이루어지기 전에 미리 되어야 한다.
받는 입장: 상대방의 Track이 감지되면 HTML Video Element에 연결해서 보여준다.

- *Serialized Json/Text Data*

주는 입장: RTCPeerConnection의 createDataChannel을 통해 특정 이름의 data 전달 채널을 개설할 수 있다. 이 또한 Signaling을 통해 connection이 이루어지기 전에 이루어져야 한다.
받는 입장: RTCPeerConnection의 onDataChannel이 이루어진 뒤에 변수에 넣어서 다시 랜더링한다.

(d) WebRTC on Online Test Platform (Cheating Detection)

① *Fetching*

Fetching 단계에서는 getDisplayMedia를 통해 화면공유, getUserMedia를 통해 웹 캠 화면을 특정 변수로 대입할 수 있다. 이를 통해 HTML Video element로 학생 테스트 페이지에서 웹 캠을 나타나게 하고, addTracks, onTracks를 통해 P2P로 연결된 상대에게 영상을 보낼 수 있다.

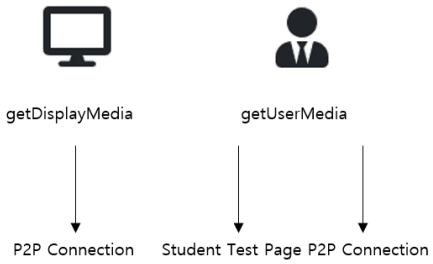


Figure 29. Display & User Fetching

② *Signaling + Connection*

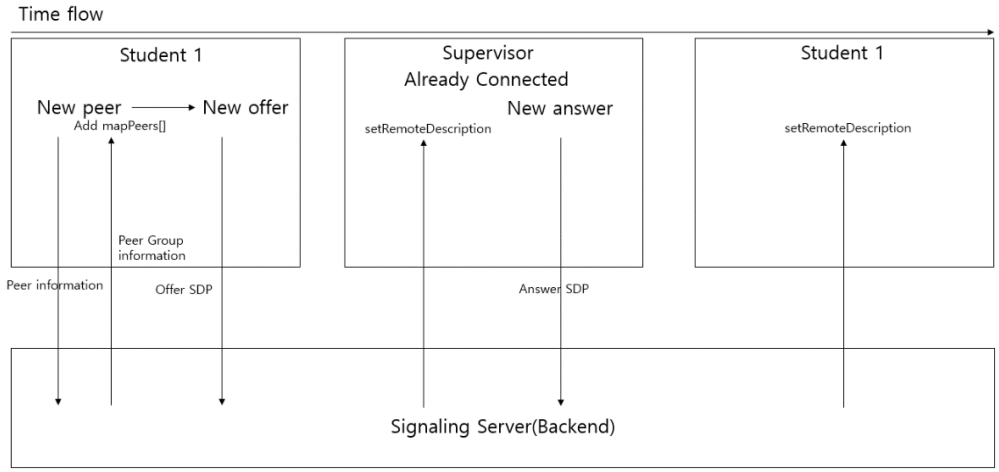


Figure 30. Signaling Process with Time Flow

관리자가 이미 연결되어 있다고 가정할 때, Student 1이 접속한다고 해보자. Student 1은 New peer을 통해 Signaling Server로 피어 정보를 전송하면 접속되어 있는 피어 정보들을 받아온다. 이것을 mapPeers에 더하여서 이미 있는 참가자(Supervisor)를 확인한 뒤, New offer을 통해 Offer SDP를 보낸다. 그렇게 되면 감독관은 setRemoteDescription을 수행하고 New answer을 통해 Answer SDP를 보내게 되면 Student 1은 setRemoteDescription을 수행하면서 서로가 P2P연결을 성공하게 된다.

③ *Communication*

Communication 단계에서는 일반적인 화상 채팅 어플리케이션이라면 서로가 서로에게 비디오를 보내면 되지만, 이 플랫폼에서는 모두가 그럴 필요가 없다. 학생이 관리자에게 웹 캠 화면과 Screen 화면을 보내기만 하면 된다. 하지만 Chat은 서로 상호작용할 수 있게 해야 하므로, 학생은 영상을 전송하기 위한 addTrack, 직렬화 된 데이터를 전송하기 위한 createDataChannel, 직렬화 된 데이터를 받기 위한 onDataChannel이 있어야 하고, 관리자는 영상을 받기 위한 onTrack, 직렬화 된 데이터를 전송하기 위한 createDataChannel, 직렬화 된 데이터를 받기 위한 onDataChannel이 있어야 한다.

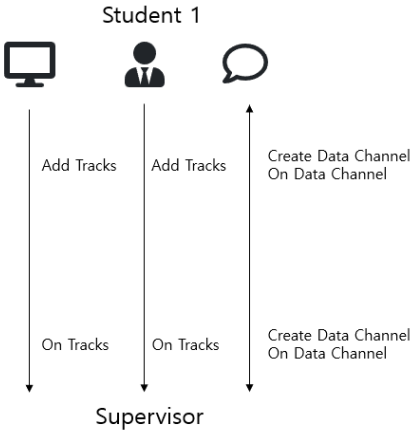


Figure 31. Communication in Cheating Detection

ii. *Supervisor Test Page*

(a) *Student Control*

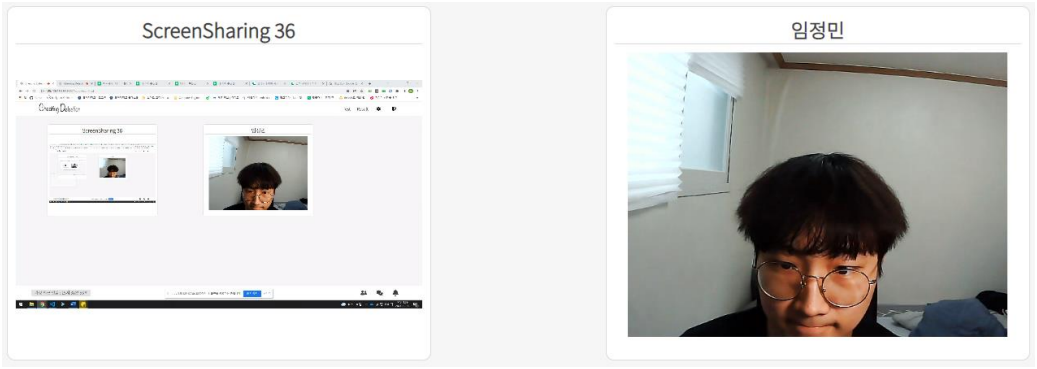


Figure 32. Student Control

학생들이 Pre-Check하는 과정에서 화면공유를 하고 테스트 화면으로 가기 때문에 화면공유 페이지가 좌측, 웹 캠 화면이 우측에 위치한다. 학생들이 접속하지 않았을 때는 박스가 생성되지 않고, 화면공유 버튼을 클릭하거나, 학생이 테스트 화면에 접속하면 CreateElement를 통해 박스가 생성된다. 학생이 시험을 칠 때 시선이 이탈하면 학생 박스의 뒷부분이 점점 붉게 변하게 되고 이는 관리자가 시선이탈을 습관적으로 하는 학생을 주목하기 자연스럽다.

(b) *Participant*

우측 하단에 참가자, 채팅, 알람 아이콘이 위치하는데, 이를 클릭하면 생성되고 재 클릭하거나 다른 아이콘 클릭 시 사라지는 형태이다. 참가자 목록 아이콘은 현재 참가하고 있는 인원을 나타낸다.

(c) *Chat*

채팅 아이콘은 관리자가 학생에게 시선 이탈을 경고 줄 때 사용하거나, 공지사항 등을 안내하기 위해 생성하였다. 채팅은 데이터 채널을 통해서 이루어진다. Data Channel에 메시지

를 보내면 다른 사용자들에게 전부 보내는 형태이다.

(d) Alarm

알람 시스템은 React Table 라이브러리를 이용하였는데, 검색 기능과 정렬기능을 포함하고 있어 특정 학생의 알람 발생만을 볼 수 있다.

Figure 33. Participant & Chatting & Alarm Log

iii. Student Pre-Check Page

(a) Pre-Check Navigation Bar

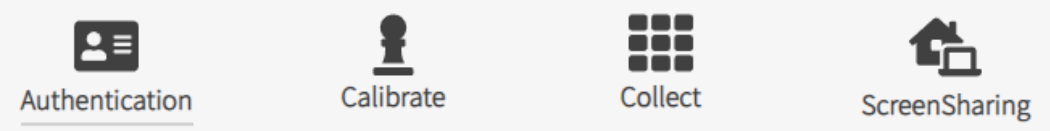


Figure 34. Pre-Check Navigation Bar

Pre-Check의 과정의 어디에 위치하였는 지를 확인하기 위해 만든 네비게이션 바로 전체 네비게이션과 마찬가지로 현재 RL을 인식하여 밑줄 표시를 하였다.

(b) Authentication

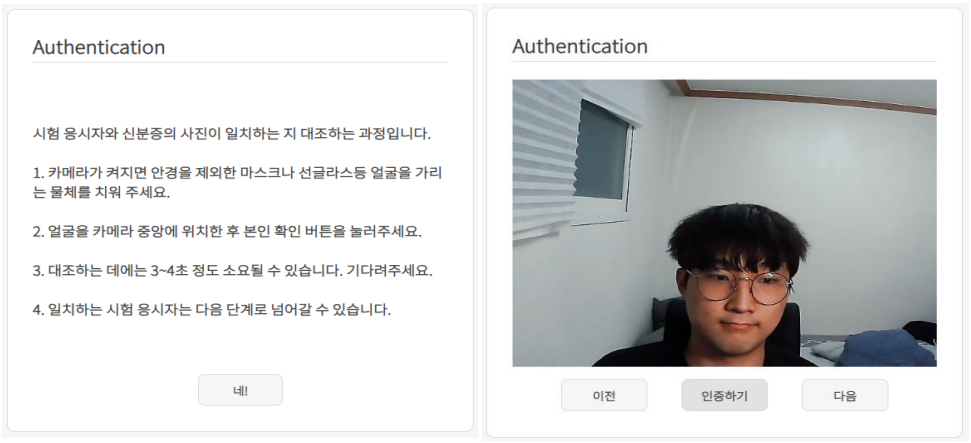


Figure 35. Authentication Process

네비게이션에서 Test페이지를 클릭하면 제일 처음 나타나는 페이지로 대리시험을 방지하기 위해 있는 페이지이다. 처음에는 Authentication의 과정 설명을 하고 네! 버튼을 누르면 자신의 웹 캠이 나타나게 된다. 이는 자신의 신분증에 있는 본인 사진과 비교하는 과정으로 인증하기를 누르면 누를 때 이미지 프레임은 서버로 보내지고 Authentication Model에서 검증 과정을 거친 뒤 인증완료, 다른 사람이 부정행위를 하려고 하는 지, 다른 사람이 신분증 사진을 통해 부정행위를 하려고 하는 지 비교한 뒤 사용자에게 Alert를 보낸다.

(c) Calibrate, Collect

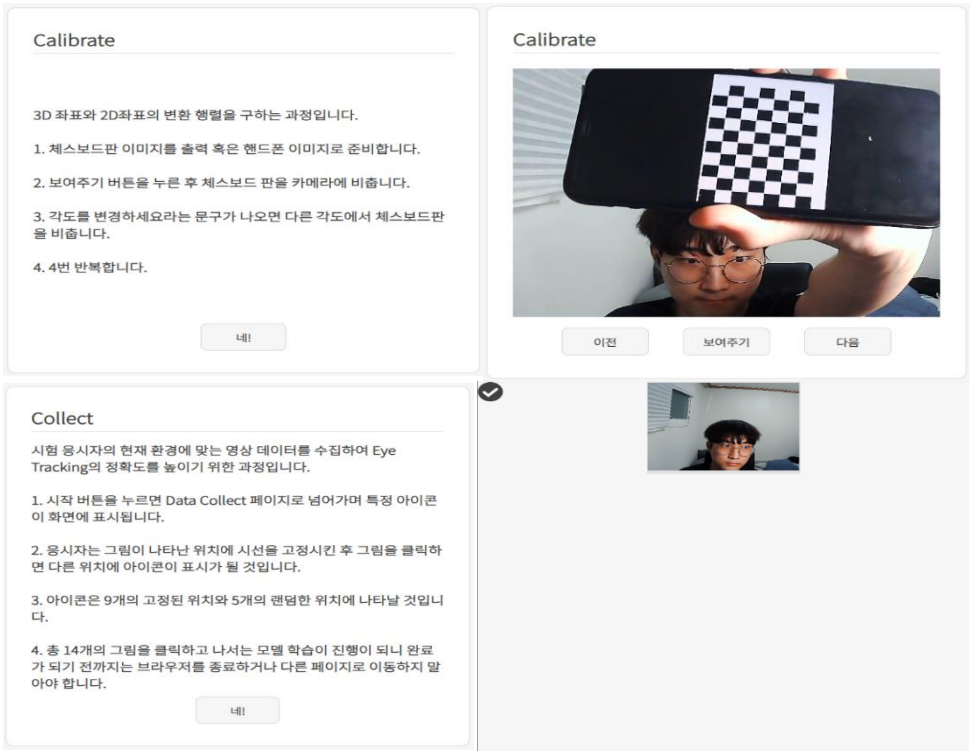


Figure 36. Calibrate & Collect Process

Calibrate, Collect 과정은 카메라 시선 이탈을 감지하기 위해 준비하는 단계이다. Calibrate 과정은 웹 캠의 내부 파라미터를 구하기 위하여 웹 캠에 체스보드 판을 위치하여서 Eye

Tracking 모델이 인식하고 파라미터를 조정하게 하는 것이고, Collect 과정은 사용자의 현재 환경에 맞는 영상 데이터를 수집하여 Fine Tuning을 하는 것이다. 동일한 박스 형태에서 웹캠과 안내 문구가 나타나는 형태이고, 마지막 Collect 과정은 사용자의 실제 브라우저 화면을 15*15 Grid Display로 분할한 다음, (1,1)Grid에 아이콘이 위치하고 (1,8)에 아이콘이 위치하게 하는 방식으로 아이콘을 움직였다.

(d) Screen Sharing

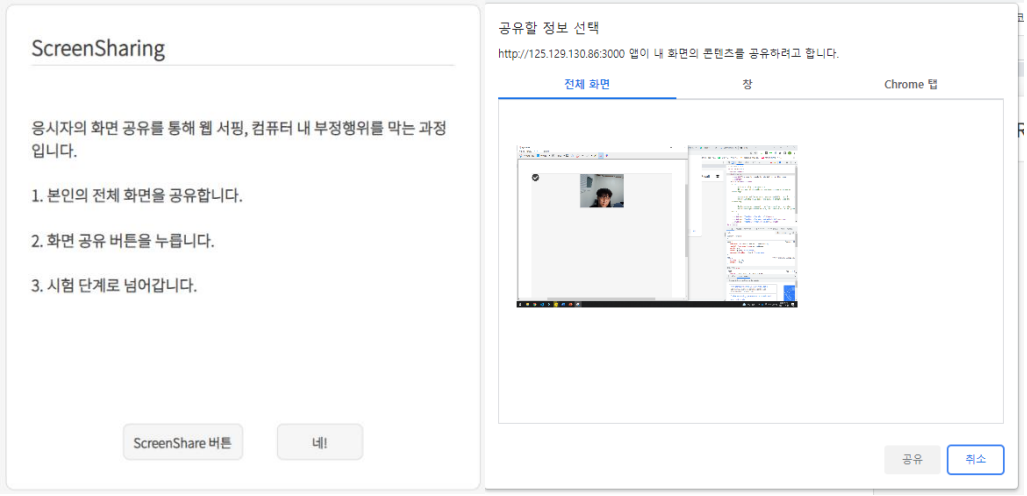


Figure 37. Screen Sharing Process

화면공유 페이지는 WebRTC에서 제공하는 API인 getDisplayMedia를 통해 전체화면, 창 혹은 Chrome 탭을 선택하여 공유할 수 있고, ScreenShare버튼을 누르게 되면 연결된 Track에 자신의 공유할 화면을 실시간으로 전송하게 된다.

iv. Student Test Page

학생 테스트 페이지에서는 웹 캠을 Track를 통해 관리자에게 전송하고, 좌측에는 시험 문제, 우측에는 Grid display로 자신의 웹 캠, 답안지, 채팅 3가지를 나타낸다. 우측에 있는 박스는 fixed position을 통해 시험 문제를 내리면 박스도 동시에 움직이게 된다.

C. Backend

i. WebSocket on Online Test Platform (Cheating Detection)

(a) WebSocket on Online Test Platform

Authentication, Calibrate, Collect, Screen Sharing, Test Page에 모두 소통을 각각의 다른 WebSocket으로 하였다. 그 이유는 Pre-Check 과정과 테스트 과정 모두 웹 캠을 이용하고 동영상 프레임을 서버에 전송하는 경우가 대부분이기 때문에 WebSocket을 이용하여 비용을 절감하였다. Authentication의 경우 WebSocket 연결 후 인증하기 버튼을 누를 때 send로 웹캠 이미지 프레임을 전송한다. Calibrate는 서버에 웹 캠 자체를 전송한 뒤, 체스판을 Eye Tracking Model이 인식하게 되면 4회 반복 후 전송을 멈춘다. Collect의 경우 아이콘을 클릭할 때 기준으로 그 이미지 프레임과 0.03초 주기로 4장의 이미지 프레임을 추가로 보낸다. 그렇게 9개의 정해진 위치의 아이콘을 클릭하고 5개의 랜덤 한 위치의 아이콘을 클릭하여 총 70장의 이미지 프레임을 Eye Tracking Model로 전송하고 연결을 끊는다. Screen Sharing은

테스트와 같은 WebSocket을 사용하는데, P2P Connection을 하기 위한 Signaling Server 역할을 하게 된다.

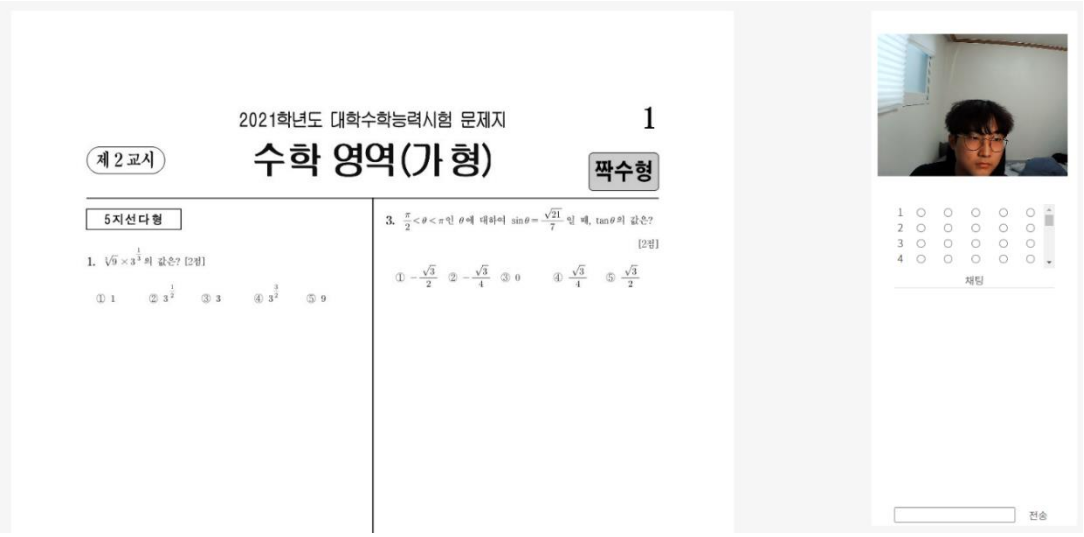


Figure 38. Student Test Page

(b) WebSocket on Eye Tracking

P2P Connection과 별개로 학생들은 시선 좌표를 Eye Tracking Model로부터 받아올 필요가 있다. 이 때 Supervisor가 직접 Eye Tracking Model에 접근하여 좌표를 받아오는 것이 아닌, 각각의 학생이 좌표를 받아온다. 그 이유는 한 포트에서 WebSocket을 이중으로 접근하는 것이 과정이 복잡하고 비효율적이기 때문이고, 각각의 학생이 이미지 프레임을 보내고 좌표를 받은 다음, 기존에 연결되어 있는 P2P Connection을 통하여 웹 캠, 컴퓨터 화면, 채팅을 보내고 있기 때문에 그곳에서 좌표를 보내는 것이 더욱 효율적이라 판단하였다.

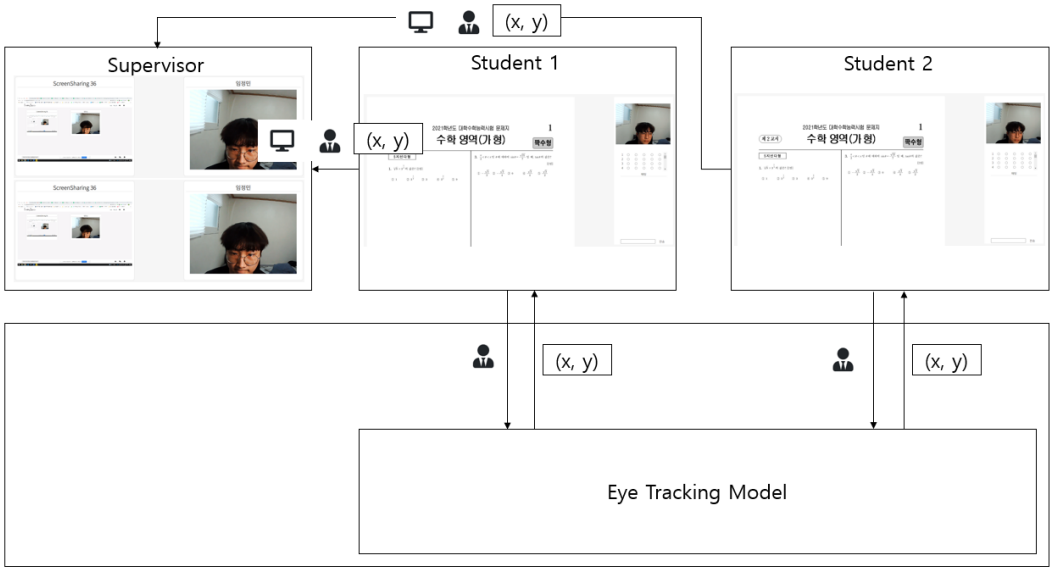


Figure 39. WebSocket on Eye Tracking Process

6 CONCLUSION

이 프로젝트에서는 실시간 온라인 테스트 플랫폼에 두 가지의 모델을 이용하여 세 가지의 부정행위를 탐지하였다. authentication 모델의 anti-spoofing 과 verification 을 통해서 본인 확인 과정을 자동화함으로써 시험자 신원 확인의 번거로움을 줄이며 대리 시험을 방지할 수 있고, Head Pose Estimation 과 Eye Tracking model 을 통하여 사용자의 얼굴 움직임과 시선을 탐지했고, WebRTC 의 Fetching, Signaling, Connection, Communication 과정을 통하여 영상/음성/텍스트를 사용자끼리 연결하고 관리자가 학생의 영상, 채팅을 받을 수 있게 하였다. 또한 WebRTC 의 Fetching 과정에서 추가로 제공하고 있는 getDisplayMedia 를 통하여 학생이 관리자에게 화면공유를 하여 웹 서핑을 이용한 부정행위를 막는다.

우리는 이러한 부정행위 탐지를 통해 온라인 시험의 공정성을 높이고 코로나 사태의 해결에 기여하기를 바란다. 더 나아가서 코로나 문제를 해결하기 위한 우리의 도약이 코로나 사태가 마무리된 이후에도 온라인 시험의 편의성을 높이고 시험 치는 방법의 고려대상 중 하나가 되기를 빈다.

7 REFERENCE

Eye-Tracking:

S. Park, S. D. Mello, P. Molchanov, U. Iqbal, O. Hilliges, and J. Kautz, “Few-shot adaptive gaze estimation,” in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 9368–9377.

Authentication:

SCHROFF, Florian; KALENICHENKO, Dmitry; PHILBIN, James. Facenet: A unified embedding for face recognition and clustering. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2015. p. 815-823.

X. Yang et al., "Face Anti-Spoofing: Model Matters, so Does Data," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 3502-3511, doi: 10.1109/CVPR.2019.00362.

https://sefiks.com/2020/05/22/fine-tuning-the-threshold-in-face-recognition/

https://chalearnlap.cvc.uab.cat/challenge/33/track/33/metrics/

Camera Calibration

https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html

https://m.cafe.daum.net/smhan/e9OL/8

Head Pose Estimation

https://towardsdatascience.com/head-pose-estimation-using-python-d165d3541600

OpenCV

https://ko.wikipedia.org/wiki/OpenCV

https://sefiks.com/2020/08/25/deep-face-detection-with-opencv-in-python/

Tesseract OCR

https://ko.wikipedia.org/wiki/%ED%85%8C%EC%84%9C%EB%9E%99%ED%8A%B8

WebRTC

http://jaynewho.com/post/36