DAFTAR PUSTAKA

Adani, M. R. (2020, Desember 20). *Pengenalan Apa Itu Website Beserta Fungsi, Manfaat dan Cara Membuatnya*. Diambil kembali dari Sekawan Media: www.sekawanmedia.co.id/

Afian, A. (2019, November 6). *Berkenalan dengan Socket.io*. Diambil kembali dari Afief Afian: afiefafian95.medium.com/

APPKEY. (2020, Oktober 16). *Web Aplikasi : Pengertian dan Keunggulannya*. Diambil kembali dari WEBAPP: https://appkey.id/

Arfienda, P. (2018, Februari 25). *Behind Data Science: Bagaimana Cara Kerja Facial Recognition?* Diambil kembali dari Algoritma: https://algorit.ma/

Brownlee, J. (2019, Mei 31). *A Gentle Introduction to Deep Learning for Face Recognition*. Diambil kembali dari Machine Learning Mastery: machinelearningmastery.com/

Darsiwan. (2016, Agustus 2). *Apa itu WebSocket*. Diambil kembali dari CODEPOLITAN: codepolitan.com

Dewaweb Team. (2021, Januari 10). *Jaringan Komputer: Pengertian, Topologi, dan Jenisnya*. Diambil kembali dari Dewaweb: www.dewaweb.com/

Efendi, J. (2017). Real Time Face Recognition using Eigenface and Viola-Jones Face Detector. *International Journal on Informatics Visualization*, 16-22.

Ermayulis, S. (2020, Agustus 23). *Penerapan Sistem Pembelajaran Daring dan Luring Di Tengah Pandemi COVID-19*. Diambil kembali dari STIT Al-Kifayah Riau: https://www.stit-alkifayahriau.ac.id/

Fadlil, A., Riadi, I., & Aji, S. (2017). Pengembangan SIstem Pengaman Jaringan Komputer Berdasarkan Analisis Forensik Jaringan. *Ilmu Teknik Elektro Komputer dan Informatika*, 8.

Fajrin, R. (2017). Pengembangan Sistem Informasi Geografis Berbasis Node.js Untuk Pemetaan Mesin dan Tracking Engineer dengan Pemanfaatan Geolocation pada PT IBM Indonesia. *Informatika*, 40-47.

Fauzan, A. (2019, Oktober 9). *Mengukur Jarak Euclidean: Teori dan Implementasi Menggunakan Java*. Diambil kembali dari Kita Informatika: www.kitainformatika.com/

Firebase Team. (t.thn.). *Dokumentasi Firebase*. Diambil kembali dari Firebase: firebase.google.com/docs/

Gheytasi, M. (2015). The effect of smartphone on the reading comprehension. *Procedia - Social and Behavioral Sciences*, 6.

Gifari, J., Widya, A., & Yovita. (2020, November 18). *Belajar Data Science : Apa yang dimaksud dengan Tensorflow dan Bagaimana Penggunaannya?* Diambil kembali dari DQLab: dqlab.id/

Irei, A. (2020, Agustus). *Real-Time Communications (RTC)*. Diambil kembali dari SearchUnifiedCommunications: searchunifiedcommunications.techtarget.com/

Khawas, C. (2018). Application of Firebase in Android App Development-A. *International Journal of Computer Applications*, 49-53.

Khoiriyah, N. L. (2018). Rancang Bangun Sistem Presensi Online Berbasis Granted Validitas Data.

Kurniawan, L. M. (2014). Metode Face Recognition untuk Identifikasi Personil Berdasar Citra Wajah bagi Kebutuhan Presensi Online Universitas Negeri Semarang. *Scientific Journal of Informatics*, 210-220.

Lina, Q. (2019, Januari 2). *Apa itu Convolutional Neural Network?* Diambil kembali dari Medium: medium.com

Martins, M. d. (2015). How to effectively integrate technology in the foreign language. *Procedia - Social and Behavioral Sciences*, 8.

Mulyani, Y. (2020). Penerapan Absensi Online Berbasis Android pada Peningkatan Kedisiplinan Dan Kinerja Guru Pegawai Negeri Sipil Pada Bidang PAI.

Nayyef, Z. T., Amer, S. F., & Hussain, Z. (2018). Peer to Peer Multimedia Real-Time Communication System based on WebRTC Technology. *International Journal of Engineering & Technology*, 125-130.

NKD, F. (2019, Mei 10). *Web App VS Website – Apa Saja Perbedaan Keduanya?* Diambil kembali dari Logique: https://www.logique.co.id/

Nordin, N. (2020). A Web-Based Mobile Attendance System with Facial Recognition Feature. *International Journal of Interactive Mobile Technologies*, 193-202.

Nurfita, R. D. (2018). Implementasi Deep Learning Berbasis Tensorflow Untuk Pengenalan Sidik Jari. *Jurnal Teknik Elektro*, 22-27.

Pratama, R. R. (2020). Analisis Model Machine Learning Terhadap Pengenalan Aktifitas Manusia. *Manajemen, Teknik Informatika & Rekayasa Komputer*, 10.

Ramadhan, F. E. (2020). *Penerapan Image Classification Dengan Pre-Trained Model MobileNet Dalam Client-Side Machine Learning.* Jakarta: Fakultas Sains Dan Teknologi, Universitas Islam Negeri Syarif Hidayatullah.

Rohman, A., & B, I. Y. (2016). Aplikasi Asisten Praktikum Menggunakan NodeJS dan Database MongoDB (Studi Kasus Lab STMIK AKAKOM). *Seminar Riset Teknologi Informasi*, 73-82.

Rustan, M. R. (2019). *Rancang Bangun Sistem Absensi Mahasiswa Menggunakan Sensor RFID Berbasis Website.* Makassar: UIN Alauddin.

Sanadi, E. A. (2018). Pemanfaatan Realtime Database di Platform Firebase Pada Aplikasi E-Tourism Kabupaten Nabire. *Penelitian Enjiniring, Fakultas Teknik*, 20-26.

Sandoval, C. G. (2019). *Multiple Face Detection and Recognition System Design Applying Deep Learning in Web Browsers using JavaScript.* Fayetteville: University of Arkansas.

Silvia. (2019, Juni 1). *What Are WebSockets?* Diambil kembali dari Jetorbit: www.jetorbit.com

Smilkov, D., & Thorat, N. (2019). TensorFlow.js: Machine Learning For The Web And Beyond. *System Modelling Language Conference.* California: arXiv.

Sofyana, L. (2019). Pembelajaran Daring Kombinasi Berbasis Whatsapp Pada Kelas Karyawan Prodi Teknik Informatika Universitas PGRI Madiun. *Jurnal Nasional Pendidikan Teknik Informatika*, 6.

Somya, R. (2018). Perancangan Aplikasi Chatting Berbasis Web di PT. Pura Barutama Kudus Menggunakan Socket.IO dan Framework Foundation. *Ilmu Komputer dan Informatika*, 8-15.

Suprianto, D. (2013). Sistem Pengenalan Wajah Secara Real-Time dengan Adaboost, Eigenface PCA & MySQL. *Electrical Power, Electronics, Communications, Controls and Informatics Seminar*, 6.

LAMPIRAN

```
(async () => {

    const faceVideo = document.querySelector('#face-video');

    const auth = (await import("../data/auth-helper"))
        .default.init();
    const db = (await import("../data/database-helper"))
        .default.init();
    const firestore = (await import("../data/firestore-helper"))
        .default.init();

    const showFaceLocation = true;
    let myDescriptors = [];
    let passPhotoDetection;

    const canvas = document.querySelector('#face-canvas');
    let USER_ID;

    auth.onAuthStateChanged(user => {
        if (user) {
            console.log('user logged in: ', user.uid);
            USER_ID = user.uid;

            showAppLoading(true, "Preparing Camera...");

            navigator.mediaDevices.getUserMedia({
                video: true, audio: false
            }).then(stream => {
                console.log('media berhasil!');
                faceVideo.srcObject = stream;
                main();
            }, err => {
                console.log(err);
                alert("Failed to access your camera! It may be used by other
application!")
            });
        } else {
            console.log('no user logged in');
            redirectToLogin();
        }
    });

    const main = async () => {
        try {
            showAppLoading(true, "Loading Models...");
```

```
            console.log('start loading models ...');
            await faceapi.loadSsdMobilenetv1Model('/models');
            await faceapi.loadFaceLandmarkModel('/models');
            await faceapi.loadFaceRecognitionModel('/models');
            console.log('loading models completed!');

            showAppLoading(true, "Initializing Detector...");
            passPhotoDetection = await firstDetection();
            showAppLoading(false);
            snapshotFaceOri();

        } catch (error) {
            showAppLoading(false);
            console.log(error);
            if (error === "passport photo undefined") {
                console.log("tidak ada wajah");
            }
        }
    }

    const firstDetection = async () => {
        let detection;
        let error;

        try {
            const passImageElement = document.createElement("img");
            passImageElement.src = (await
firestore.getUser(USER_ID))["passPhotoUrl"];
            passImageElement.crossOrigin = "anonymous";
            console.log('start first detection ... ');
            const start = new Date().getTime();
            detection = await faceapi
                .detectSingleFace(passImageElement)
                .withFaceLandmarks()
                .withFaceDescriptor();
            const finish = new Date().getTime();
            console.log('first detection complete!', finish - start, 'ms');
            console.log("passDetection:", detection);
            if (!detection) {
                throw "passport photo undefined";
            }
        } catch (e) {
            error = e;
        }

        return new Promise((resolve, reject) => {
            if (error) reject(error);
            else resolve(detection);
```

```
        });
    }

    const snapshotFaceOri = async () => {
        console.log('snapshot ori face ... ');
        const start = new Date().getTime();

        const newDetection = await faceapi
            .detectSingleFace(faceVideo)
            .withFaceLandmarks()
            .withFaceDescriptor();

        const finish = new Date().getTime();
        console.log('detection duration:', finish - start, 'ms');

        if (!newDetection) {
            console.log('Tidak ada wajah terdeteksi');

            snapshotFaceOri();
        }
        else {
            if (newDetection.detection.score >= 0.995)
                myDescriptors.push(newDetection.descriptor);

            if (myDescriptors.length < 3) {
                snapshotFaceOri();
            } else {
                console.log('face-register complete!', myDescriptors);

                const labeledFaceDescriptors = createLabeledFaceDescriptors({
                    label: USER_ID,
                    descriptors: myDescriptors
                });
                matchMyDescriptorsWithPassPhoto(labeledFaceDescriptors);
            }
        }

        if (showFaceLocation) {
            let box;
            let label = '';
            canvas.width = faceVideo.clientWidth;
            canvas.height = faceVideo.clientHeight;

            if (newDetection) {
                const detectionsForSize = faceapi.resizeResults(newDetection,
{
                    width: faceVideo.clientWidth,
                    height: faceVideo.clientHeight
```

```
                });
                box = detectionsForSize.detection.box;
                label = (newDetection.detection.score*100).toFixed(1)+"%";
            }
            console.log('BOX:', box);
            const drawBox = new faceapi.draw.DrawBox(box, { label: label });
            drawBox.draw(canvas);
        }
    }

    const createLabeledFaceDescriptors = ({ label, descriptors }) => {
        return new faceapi.LabeledFaceDescriptors(
            label,
            descriptors
        );
    }

    const matchMyDescriptorsWithPassPhoto = async (labeledFaceDescriptors) =>
{
        const faceMatcher = new faceapi.FaceMatcher(labeledFaceDescriptors,
0.5);
        const bestMatch =
faceMatcher.findBestMatch(passPhotoDetection.descriptor);
        const distance = bestMatch.distance * 100;
        console.log("Distance: ", distance);
        console.log("Label: ", bestMatch.label);
        if (bestMatch.label === USER_ID) {
            console.log("UID:", USER_ID);
            saveDescriptorsToDb(labeledFaceDescriptors);
        } else {
            showFaceRegisterFailedDialog();
            removeFaceRegisterContainer();
            console.log("Wajah tidak cocok dengan passport photo!");
        }
    }

    const saveDescriptorsToDb = (labeledFaceDescriptors) => {
        db.setDescriptors(USER_ID, labeledFaceDescriptors.toJSON())
            .then(() => {
                addFaceRegisterCompleted();
                removeFaceRegisterContainer();
            })
            .catch(error => console.log(error.message));
    }

    const showFaceRegisterFailedDialog = () => {
        const appDialog = document.querySelector("#app-dialog");
        appDialog.hidden = false;
```

```
        appDialog.querySelector("h2").innerText = `The detected face failed to
be recognized due to mismatch with the passport photo!`;

        document.querySelector("#app-dialog button").addEventListener("click",
() => {
            window.location.href = "/face-register.html";
        });
    }

    const addFaceRegisterCompleted = () => {
        document.querySelector(".box-shadow-container").innerHTML += `
            <div class="center-text" style="margin-top: 32px;">
                <h1>
                    <span class="material-icons text-success" style="font-
size: 1.7em">
                        how_to_reg
                    </span>
                </h1>
                <h2>Your face has succesfully registered</h2>
                <a href="room.html" class="btn btn-primary">Go to Home
Page</a>
            </div>
        `;
    }

    const removeFaceRegisterContainer = () => {
        const faceRegisterContainer = document.querySelector("#face-register-
container");
        faceVideo.srcObject.getTracks()[0].stop();
        faceRegisterContainer.remove();
    }

    const showAppLoading = (state = true, dialogMsg = "") => {
        const appLoading = document.querySelector("#app-loading");
        appLoading.hidden = !state;
        appLoading.querySelector("chase-dot").dialogMsg = dialogMsg;
    }

    const redirectToLogin = () => {
        window.location.href = "/login.html";
    }

})();
```

```
(async () => {

    const localVideo = document.querySelector('#local-video');
    const btnJoin = document.querySelector('#btn-join');
    const inputRoomId = document.querySelector('#input-roomid');
    const joinContainer = document.querySelector('#joinroom-container');
    const statusIndicator = document.querySelector("face-status-indicator");
    const appLoading = document.querySelector("#app-loading");
    const contentLoadingContainer = document.querySelector("#content-loading-
container");
    const contentLoading = document.querySelector("#content-loading");
    const passPhotoElement = document.querySelector("#passphoto");
    const nicknameElement = document.querySelector("#nickname");
    const joinroomBar = document.querySelector("#joinroom-bar");
    const idClassroomIndicator = document.querySelector("#id-classroom-
indicator");
    const roomNotFoundModal = new
bootstrap.Modal(document.querySelector("#room-not-found-modal"));
    const faceNotRegisteredModal = new
bootstrap.Modal(document.querySelector("#face-not-registered-modal"));

    const auth = (await import("../data/auth-helper"))
        .default.init();
    const firestore = (await import("../data/firestore-helper"))
        .default.init();
    const rtdb = (await import("../data/database-helper"))
        .default.init();

    let USER_ID;
    let ROOM_ID;
    let nickname;
    let avatarUrl;
    let role;
    let faceDescriptors;
    let classroomStartTime;
    let classroomEndTime;

    const distanceThreshold = 0.5;
    const delayAwayDetection = 15000;

    let firstSessionAway = 0;
    let isAwayFirstSession = true;

    let recognizingInterval;
```

```
    auth.onAuthStateChanged(async user => {
        if (user) {
            USER_ID = user.uid;
            await init();
            showAppLoading(false);
            getUserMedia();
        } else {
            redirectToLogin();
        }
    });

    window.addEventListener("load", () => {
        const currentUrl = new URL(window.location.href);
        ROOM_ID = currentUrl.searchParams.get("id");
        inputRoomId.value = ROOM_ID;
    });

    const init = async () => {
        try {
            btnJoin.addEventListener('click', onJoinClassroom);

            const user = await firestore.getUser(USER_ID);
            avatarUrl = user["passPhotoUrl"];
            nickname = user["name"];
            if (!nickname) nickname = "Anonymous";
            role = user["role"];
            if (role === "teacher") {
                addCreateClassroomElement();
            }
            passPhotoElement.src = avatarUrl;
            nicknameElement.innerText = nickname.split(" ").slice(-1).join();
            console.log("Nickname:", nickname);
        } catch (error) {
            console.log(error);
            alert("Ups... Something went wrong causes the app is not loaded
perfectly!")
        }
    }

    const getFaceDescriptors = async () => {
        const labeledFaceDescriptors = await rtdb.getDescriptors(USER_ID);
        return
faceapi.LabeledFaceDescriptors.fromJSON(labeledFaceDescriptors);
    }

    const getUserMedia = async () => {
        const devices = await navigator.mediaDevices.enumerateDevices();
```

```
        const cameraDevices = devices.filter(device => device.kind ===
"videoinput");
        if (cameraDevices[0].label === "") {
            showRequestCameraAccessPermission();
        }
        navigator.mediaDevices.getUserMedia({
            video: true, audio: false
        }).then(localStream => {
            showRequestCameraAccessPermission(false);
            showJoinContainer(true);
            showNavDrawer(true);
            localVideo.srcObject = localStream;
        }, err => {
            console.log("media gagal", err);
            alert("Failed to access your camera! It may be used by other
application!");
        });
    }

    const onJoinClassroom = async () => {
        ROOM_ID = inputRoomId.value;
        console.log("onJoinClassroom");
        if (!ROOM_ID) {
            return;
        }

        showContentLoading(true, "Joining Room...");
        try {
            if (role === "student") {
                faceDescriptors = await getFaceDescriptors();
            }
            const room = await firestore.getClassroom(ROOM_ID);
            console.log("classroom", room);
            showContentLoading(false);
            classroomStartTime = room.startTime;
            const duration = room.duration;
            classroomEndTime = classroomStartTime + duration;
            const currentTime = new Date().getTime();
            if (currentTime >= classroomEndTime && role === "teacher") {
                alert("This classroom meeting has ended. You can still use it,
" +
                    "but student attendance will no longer be detected
automatically.");
            }
            joinClassroom(room);
        } catch (error) {
            console.log(error);
            showContentLoading(false);
```

```
            if (error.code === "room not found") {
                roomNotFoundModal.show();
            }
            else if (error === "no face descriptors found!") {
                faceNotRegisteredModal.show();
            }
            else {
                alert("Ups, Something is wrong... Please try again later!");
            }
        }
    }

    const transactClassroomStudents = (ROOM_ID, USER_ID) => {
        return firestore.transactionClassroom(ROOM_ID, (transaction,
classroomSnapshot) => {
            const classroomData = classroomSnapshot.data();
            let students = classroomData["students"];
            if (students) {
                if (!students.includes(USER_ID))
                    students.push(USER_ID);
            } else {
                students = [USER_ID];
            }
            transaction.update(classroomSnapshot.ref, { students });
        }).then(() => {
            console.log("Transaction succesfully committed!");
        }).catch(error => {
            console.log("Transaction failed:", error);
        });
    }

    const joinClassroom = (room) => {
        showJoinContainer(false);
        showNavDrawer(false);
        startVC(room);
    }

    const startVC = (room) => {
        const domain = "meet.jit.si";
        const options = {
            roomName: room.id,
            parentNode: document.querySelector('#facevideo-container'),
        }
        const api = new JitsiMeetExternalAPI(domain, options);
        idClassroomIndicator.querySelector("input").value = ROOM_ID;
        showIdClassroomIndicator();
        let hideIndicatorTimeout;
```

```
        api.executeCommand('displayName', nickname);
        api.executeCommand('avatarUrl', avatarUrl);
        api.executeCommand('subject', room.name);
        api.addListener('readyToClose', () => { // Jitsi meet closed
            console.log('Event:', 'Ready To Close');
            api.getIFrame().remove();
            showJoinContainer();
            showNavDrawer();
        });
        api.addListener("videoConferenceJoined", () => {
            console.log("Event: Video Conference Joined!");
            if (role === "student") {
                const transResult = transactClassroomStudents(ROOM_ID,
USER_ID);
                console.log("transResult:", transResult);
                showContentLoading(true, "Loading Models...");
                showStatusIndicator();
                doFaceRecognition();
            }
            showIdClassroomIndicator(true);
            statusIndicator.classList.add("hidden-for-status-indicator");

            hideIndicatorTimeout = setTimeout(() => {
                showIdClassroomIndicator(false);
                statusIndicator.classList.remove("hidden-for-status-
indicator");

                const setupIndicatorDisplayOnMouse = () => {
                    showIdClassroomIndicator(true);
                    statusIndicator.classList.add("hidden-for-status-
indicator");
                    if (hideIndicatorTimeout)
clearTimeout(hideIndicatorTimeout);
                    hideIndicatorTimeout = setTimeout(() => {
                        showIdClassroomIndicator(false);
                        statusIndicator.classList.remove("hidden-for-status-
indicator");
                    }, 4000);
                }
                api.addListener("mouseEnter", setupIndicatorDisplayOnMouse);
                api.addListener("mouseMove", setupIndicatorDisplayOnMouse);
            }, 8000);
        });
        api.addListener("videoConferenceLeft", () => {
            showStatusIndicator(false);
            firstSessionAway = 0;
            isAwayFirstSession = true;
            if (recognizingInterval) {
```

```
                console.log("I am leaving");
                clearInterval(recognizingInterval);
            }
        });
        api.addListener("chatUpdated", ({ isOpen, unreadCount }) => {
            if (isOpen) showStatusIndicator(false);
            else showStatusIndicator();
            console.log("chatUpdated:", isOpen);
        });
    }

    const initialDetection = async () => {
        const passImageElement = document.createElement("img");
        passImageElement.src = avatarUrl;
        passImageElement.crossOrigin = "anonymous";
        console.log('start first detection ... ');
        const start = new Date().getTime();
        await faceapi
            .detectSingleFace(passImageElement)
            .withFaceLandmarks()
            .withFaceDescriptor();
        const finish = new Date().getTime();
        console.log('first detection complete!', finish - start, 'ms');
    }

    const doFaceRecognition = async () => {
        console.log('loading the models ..');
        await faceapi.loadSsdMobilenetv1Model('/models');
        await faceapi.loadFaceLandmarkModel('/models');
        await faceapi.loadFaceRecognitionModel('/models');
        console.log('successfully loaded models');

        showContentLoading(true, "Initializing Detector...");
        await initialDetection();

        showContentLoading(false);
        recognizingInterval = setInterval(async () => {
            const detectStartTime = new Date().getTime();
            const detections = await faceapi.detectAllFaces(localVideo)
                .withFaceLandmarks()
                .withFaceDescriptors();
            detectFinishTime = new Date().getTime();
            console.log("detection complete:", (detectFinishTime -
detectStartTime) + "ms");

            let results = [];
            if (detections.length) {
```

```
                const faceMatcher = new faceapi.FaceMatcher(faceDescriptors,
distanceThreshold);
                let isThereMyFace = false;
                detections.forEach(detect => {
                    const bestMatch =
faceMatcher.findBestMatch(detect.descriptor);
                    const similarity = (1 - bestMatch.distance) * 100;
                    console.log("Similarity:", similarity);
                    if (bestMatch.label === USER_ID) {
                        results.push(`${nickname} ${similarity.toFixed(1)}%`);
                        isThereMyFace = true;
                    } else {
                        results.push(`Unknown ${similarity.toFixed(1)}%`);
                    }
                });

                if (isThereMyFace) {
                    onUpdatePresenceStatus('on');
                    if (!isAwayFirstSession) {
                        isAwayFirstSession = true;
                    }
                }
                else {
                    // pertama kali away
                    if (isAwayFirstSession) {
                        isAwayFirstSession = false;
                        firstSessionAway = new Date().getTime();
                    }
                    const currentSessionAway = new Date().getTime();
                    console.log(currentSessionAway - firstSessionAway);
                    if ((currentSessionAway - firstSessionAway) >=
delayAwayDetection) {
                        onUpdatePresenceStatus('away');
                    }
                    else {
                        onUpdatePresenceStatus('on');
                    }
                }
            }
            else {
                // pertama kali away
                if (isAwayFirstSession) {
                    isAwayFirstSession = false;
                    firstSessionAway = new Date().getTime();
                }
                const currentSessionAway = new Date().getTime();
                console.log(currentSessionAway - firstSessionAway);
```

```
                if ((currentSessionAway - firstSessionAway) >=
delayAwayDetection / 2) {
                    onUpdatePresenceStatus('away');
                }
                else {
                    onUpdatePresenceStatus('on');
                }
            }
        }, 1000);
    }

    const onUpdatePresenceStatus = status => {
        updateStatusIndicator(status);
        const detectStartTime = new Date().getTime();
        if (detectStartTime > classroomEndTime || detectStartTime <
classroomStartTime) {
            console.log("not started, or over!");
            return;
        }
        const newPresenceStatus = { status };
        savePresenceStatusToDb(newPresenceStatus);
    }

    const updateStatusIndicator = status => {
        if (status === "on") {
            statusIndicator.switchStatusToOn();
        } else if (status === "away") {
            statusIndicator.switchStatusToAway();
        }
    }

    const savePresenceStatusToDb = (newPresenceStatus) => {
        rtdb.transactPresentSession(ROOM_ID, USER_ID, newPresenceStatus);
    }

    const onCreateClassroom = async () => {
        const roomName = document.querySelector('#input-roomname-
create').value;
        let date = document.querySelector('#input-date-create').value;
        const time = document.querySelector('#input-time-create').value;
        let duration = document.querySelector('#input-duration-create').value;
        const unit = document.querySelector('#select-durationunit-
create').value;

        if (!date || !time) {
            alert("Date and time fields are required!");
            return;
        }
```

```
    if (unit === 'jam') {
        duration *= 3600000;
    } else if (unit === 'menit') {
        duration *= 60000;
    }

    const [year, month, day] = date.split('-');
    const [hour, minute] = time.split(':');

    date = new Date();
    date.setFullYear(year);
    date.setMonth(month - 1);
    date.setDate(day);
    date.setHours(hour);
    date.setMinutes(minute);
    date.setSeconds(0);
    date.setMilliseconds(0);

    const newRoom = {
        name: roomName,
        nameInsensitive: roomName.toUpperCase(),
        keywords: roomName.toUpperCase().split(" "),
        host: USER_ID,
        startTime: date.getTime(),
        duration: duration
    }

    try {
        showContentLoading(true, "Creating a room...");
        const newRoomData = await firestore.addThenGetClassroom(newRoom);
        ROOM_ID = newRoomData.id;
        showContentLoading(false);
        joinClassroom(newRoomData);
    } catch (error) {
        console.log(error);
    }
}

const showJoinContainer = (state = true) => {
    joinContainer.hidden = !state;
}

const showNavDrawer = (state = true) => {
    document.querySelector("navigation-drawer").hidden = !state;
}

const showStatusIndicator = (state = true) => {
```

```
        if (role !== "student") state = false;
        statusIndicator.hidden = !state;
    }

    const showRequestCameraAccessPermission = (state = true) => {
        document.querySelector("#request-access-camera-container").hidden =
!state;
    }

    const showAppLoading = (state = true) => {
        appLoading.hidden = !state;
    }

    const showContentLoading = (state = true, dialogMsg) => {
        contentLoadingContainer.hidden = !state;
        if (dialogMsg) {
            contentLoading.dialogMsg = dialogMsg;
        } else {
            contentLoading.dialogMsg = "";
        }
    }

    const showIdClassroomIndicator = (state = true) => {
        if (role === "student") return;
        if (state) idClassroomIndicator.classList.remove("hide-bottom");
        else idClassroomIndicator.classList.add("hide-bottom");
    }

    const addCreateClassroomElement = () => {
        const orElement = document.createElement("p");
        orElement.className = "caps-text";
        orElement.innerText = "Or";

        const btnCreateRoom = document.createElement('button');
        btnCreateRoom.className = "btn btn-outline-primary";
        btnCreateRoom.setAttribute("data-bs-toggle", "modal");
        btnCreateRoom.setAttribute("data-bs-target", "#create-room-modal");
        btnCreateRoom.innerText = "Create Classroom";

        joinroomBar.appendChild(orElement);
        joinroomBar.appendChild(btnCreateRoom);

        document.querySelector("#btn-create-
classroom").addEventListener("click", onCreateClassroom);
    }

    const redirectToLogin = () => {
        window.location.href = "/login.html"
```

```
    }

    const redirectToFaceRegister = () => {
        window.location.href = "/face-register.html"
    }

    /**
     * Logout handling
     */
    document.querySelector("#menu-drawer-logout")
        .addEventListener("click", async () => {
            await auth.signOut();
        });

})();
```

```
(async () => {

    const classroomName = document.querySelector("#classroom-name");
    const classroomDuration = document.querySelector("#classroom-duration");
    const classroomStart = document.querySelector("#classroom-start");
    const classroomIdIndicator = document.querySelector("#classroom-id");
    const btnOpenMeeting = document.querySelector("#btn-open-meeting");

    const auth = (await import("../data/auth-helper"))
        .default.init();
    const firestore = (await import("../data/firestore-helper"))
        .default.init();
    const rtdb = (await import("../data/database-helper"))
        .default.init();

    let idClassroom;
    let userData;

    auth.onAuthStateChanged(async user => {
        if (user) {
            const userId = user.uid;
            userData = await firestore.getUser(userId);
            const role = userData["role"];
            if (role === "student") {
                alert("Detail Page is only for teacher!");
                redirectToHome();
                return;
            }
            await main();
        } else {
            redirectToLogin();
        }
    });

    const main = async () => {
        btnOpenMeeting.addEventListener("click", onOpenMeeting);
        try {
            const currentUrl = new URL(window.location.href);
            idClassroom = currentUrl.searchParams.get("id"); // id classroom
            const classroom = await firestore.getClassroom(idClassroom);
            classroomIdIndicator.innerText = classroom.id;
            classroomName.innerText = classroom.name;
            import("../helpers/time2string-converter")
                .then(({ duration2String, time2String }) => {
```

```
                    classroomDuration.innerText =
duration2String(classroom.duration);
                    classroomStart.innerText =
time2String(classroom.startTime);
                });
            const roomAttendance = await rtdb.getAttendance(idClassroom);
            const studentsPresences = await calculatePresences(roomAttendance,
classroom);
            const attendanceTableElement = document.querySelector("presention-
table");
            attendanceTableElement.studentsPresentions = studentsPresences;
            showPresentionLoading(false);
        } catch (error) {
            showPresentionLoading(false);
            console.log("error:", error);
        }
    };

    const calculatePresences = (attendance, classroom) => {
        return new Promise(async (resolve, reject) => {
            try {
                let studentsPresenceResult = [];
                for (const studentId in attendance) {
                    let student = await firestore.getUser(studentId);
                    student["id"] = studentId;

                    let OnPresenceStatusLength = 0; // durasi status hadir
(detik)
                    const presence = attendance[studentId];
                    for (const i in presence) {
                        if (presence[i].status === "on")
                            OnPresenceStatusLength += presence[i].length;
                    };

                    const presenceInPercent = OnPresenceStatusLength * 100000
/ classroom.duration;
                    student["presentionInPercent"] = presenceInPercent;
                    studentsPresenceResult.push(student);
                }
                resolve(studentsPresenceResult);
            } catch (error) {
                reject(error);
            }
        });
    };

    const redirectToLogin = () => {
        window.location.href = "/login.html";
```

```
    }

    const redirectToHome = () => {
        window.location.href = "/room.html";
    }

    const showPresentionLoading = (state = true) => {
        const presentionLoading = document.querySelector("#presention-
loading");
        presentionLoading.hidden = !state;
    }

    const onOpenMeeting = () => {
        window.location.href = "room.html?id=" + idClassroom;
    }
})();
```

Lampiran 4. package.json

```
{
  "name": "onlineschool",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "emu": "firebase emulators:start --import=emudata --export-on-
exit=emudata",
    "host": "firebase serve --port 5000",
    "start": "nodemon -e js,html,css",
  },
  "author": "Muhammad Muflihun Naim",
  "license": "ISC",
  "dependencies": {
    "@tensorflow/tfjs-node": "^3.7.0",
    "bootstrap": "^5.0.0-beta3",
    "materialize-css": "^1.0.0-rc.2"
  },
  "devDependencies": {
    "@babel/core": "^7.13.16",
    "@babel/preset-env": "^7.13.15",
    "babel-loader": "^8.2.2",
    "clean-webpack-plugin": "^4.0.0-alpha.0",
    "codeceptjs": "^3.1.1",
    "copy-webpack-plugin": "^8.1.1",
    "css-loader": "^5.2.4",
    "firebase-admin": "^9.11.0",
    "html-webpack-plugin": "^5.3.1",
    "jasmine-core": "^3.8.0",
    "karma": "^6.3.4",
    "karma-chrome-launcher": "^3.1.0",
    "karma-jasmine": "^4.0.1",
    "karma-sourcemap-loader": "^0.3.8",
    "karma-webpack": "^5.0.0",
    "nodemon": "^2.0.7",
    "puppeteer": "^10.2.0",
    "sharp": "^0.28.3",
    "style-loader": "^2.0.0",
    "webpack": "^5.35.1",
    "webpack-cli": "^4.6.0",
    "webpack-dev-server": "^3.11.2",
    "webpack-merge": "^5.7.3"
  }
}
```