Laporan Praktikum

Mata Kuliah PBO&Pemrograman WEB

PRAKTIKUM LOGIN MULTI ROLE

Dosen Pengampu:
Wildan Aprizall Arifin, S.Pd., M.Kom.

Disusun Oleh:

Nelita Maharani

2307052

PROGRAM STUDI SISTEM INFORMASI KELAUTAN

UNIVERSITAS PENDIDIKAN INDONESIA

2024

I. PENDAHULUAN

Dalam era digital yang semakin berkembang, sistem aplikasi modern dituntut untuk memberikan akses yang aman dan terstruktur sesuai dengan kebutuhan pengguna. Salah satu implementasi penting untuk mencapai hal ini adalah dengan menyediakan fitur **login multi role**. Sistem login multi role memungkinkan pengguna dengan peran (role) yang berbeda untuk mengakses fitur dan informasi yang sesuai dengan hak akses mereka. Hal ini menjadi krusial, terutama dalam aplikasi yang melayani berbagai jenis pengguna, seperti admin, staf, dan pengguna umum. Fitur login multi role dirancang untuk meningkatkan keamanan dan efisiensi aplikasi dengan membatasi akses hanya pada fungsi yang relevan untuk setiap peran. Misalnya, seorang administrator memiliki kontrol penuh terhadap pengelolaan sistem, sementara pengguna biasa hanya dapat mengakses fitur-fitur dasar. Dengan demikian, risiko kesalahan atau penyalahgunaan data dapat diminimalisir. Selain aspek keamanan, login multi role juga memberikan pengalaman pengguna yang lebih terpersonalisasi. Setiap pengguna akan melihat antarmuka dan fitur yang disesuaikan dengan peran mereka, menciptakan navigasi yang lebih intuitif dan meningkatkan produktivitas. Oleh karena itu, penerapan login multi role dalam sebuah sistem bukan hanya sekadar opsi, tetapi menjadi kebutuhan penting dalam membangun aplikasi yang efisien, aman, dan skalabel.

II. ALAT DAN BAHAN

1. Komputer/Laptop
2. Visual Studio Code
3. Chrome

III. LANGKAH KERJA

1. Cofig/database

```
import {Sequelize} from "sequelize";

const db = new Sequelize('auth_db', 'root', '', {
    host: "localhost",
    dialect: "mysql"
});

export default db;
```

Kode ini mengelola sistem autentikasi, manajemen produk, dan pengguna dengan pendekatan berbasis peran (**Role-Based Access Control**).

2. controllers/auth

```
import User from "../models/UserModel.js";
import argon2 from "argon2";

export const Login = async (req, res) =>{
    const user = await User.findOne({
        where: {
            email: req.body.email
        }
    });
    if(!user) return res.status(404).json({msg: "User tidak ditemukan"});
    const match = await argon2.verify(user.password, req.body.password);
    if(!match) return res.status(400).json({msg: "Wrong Password"});
    req.session.userId = user.uuid;
    const uuid = user.uuid;
    const name = user.name;
    const email = user.email;
    const role = user.role;
    res.status(200).json({uuid, name, email, role});
}

export const Me = async (req, res) =>{
    if(!req.session.userId){
        return res.status(401).json({msg: "Mohon login ke akun Anda!"});
    }
    const user = await User.findOne({
        attributes:['uuid','name','email','role'],
        where: {
            uuid: req.session.userId
```

```
        }
    });
    if(!user) return res.status(404).json({msg: "User tidak
ditemukan"});
    res.status(200).json(user);
}

export const logOut = (req, res) =>{
    req.session.destroy((err)=>{
        if(err) return res.status(400).json({msg: "Tidak dapat
logout"});
        res.status(200).json({msg: "Anda telah logout"});
    });
}
```

Pada bagian autentikasi, fungsi `Login` memverifikasi kredensial pengguna dengan mencocokkan email dan password yang telah di-hash menggunakan **argon2**. Setelah login berhasil, sesi pengguna disimpan menggunakan ID unik. Fungsi `Me` digunakan untuk memeriksa status login dan mengembalikan data pengguna yang sedang aktif, sedangkan `logOut` menghapus sesi pengguna untuk logout dari sistem.

3. controllers/produk

```
import Product from "../models/ProductModel.js";
import User from "../models/UserModel.js";
import {Op} from "sequelize";

export const getProducts = async (req, res) =>{
    try {
        let response;
        if(req.role === "admin"){
            response = await Product.findAll({
                attributes:['uuid','name','price'],
                include:[{
                    model: User,
                    attributes:['name','email']
                }]
            });
        }else{
            response = await Product.findAll({
                attributes:['uuid','name','price'],
                where:{
                    userId: req.userId
                },
                include:[{
```

```
                    model: User,
                    attributes:['name','email']
                }]
            });
        }
        res.status(200).json(response);
    } catch (error) {
        res.status(500).json({msg: error.message});
    }
}

export const getProductById = async(req, res) =>{
    try {
        const product = await Product.findOne({
            where:{
                uuid: req.params.id
            }
        });
        if(!product) return res.status(404).json({msg: "Data tidak ditemukan"});
        let response;
        if(req.role === "admin"){
            response = await Product.findOne({
                attributes:['uuid','name','price'],
                where:{
                    id: product.id
                },
                include:[{
                    model: User,
                    attributes:['name','email']
                }]
            });
        }else{
            response = await Product.findOne({
                attributes:['uuid','name','price'],
                where:{
                    [Op.and]:[{id: product.id}, {userId: req.userId}]
                },
                include:[{
                    model: User,
                    attributes:['name','email']
                }]
            });
        }
```

```
        res.status(200).json(response);
    } catch (error) {
        res.status(500).json({msg: error.message});
    }
}

export const createProduct = async(req, res) =>{
    const {name, price} = req.body;
    try {
        await Product.create({
            name: name,
            price: price,
            userId: req.userId
        });
        res.status(201).json({msg: "Product Created Successfuly"});
    } catch (error) {
        res.status(500).json({msg: error.message});
    }
}

export const updateProduct = async(req, res) =>{
    try {
        const product = await Product.findOne({
            where:{
                uuid: req.params.id
            }
        });
        if(!product) return res.status(404).json({msg: "Data tidak
   ditemukan"});
        const {name, price} = req.body;
        if(req.role === "admin"){
            await Product.update({name, price},{
                      where:{
                          id: product.id
                      }
                  });
              }else{
                  if(req.userId !== product.userId) return
   res.status(403).json({msg: "Akses terlarang"});
                  await Product.update({name, price},{
                      where:{
                          [Op.and]:[{id: product.id}, {userId:
   req.userId}]
                      }
```

```
                });
            }
            res.status(200).json({msg: "Product updated
   successfuly"});
        } catch (error) {
            res.status(500).json({msg: error.message});
        }
    }

    export const deleteProduct = async(req, res) =>{
        try {
            const product = await Product.findOne({
                where:{
                    uuid: req.params.id
                }
            });
            if(!product) return res.status(404).json({msg: "Data tidak
   ditemukan"});
            const {name, price} = req.body;
            if(req.role === "admin"){
                await Product.destroy({
                    where:{
                        id: product.id
                    }
                });
            }else{
                if(req.userId !== product.userId) return
   res.status(403).json({msg: "Akses terlarang"});
                await Product.destroy({
                    where:{
                        [Op.and]:[{id: product.id}, {userId:
   req.userId}]
                    }
                });
            }
            res.status(200).json({msg: "Product deleted
   successfuly"});
        } catch (error) {
            res.status(500).json({msg: error.message});
        }
    }
```

Dalam manajemen produk, terdapat fungsi seperti `getProducts` dan `getProductById` yang mengambil data produk dengan pembatasan akses berdasarkan peran pengguna. Pengguna dengan peran **admin** dapat melihat semua produk, sementara pengguna biasa hanya dapat melihat produk

mereka sendiri. Fungsi `createProduct` memungkinkan penambahan produk baru, sementara `updateProduct` dan `deleteProduct` digunakan untuk memperbarui atau menghapus produk dengan validasi kepemilikan produk dan peran pengguna.

4. controllers/users

```
import User from "../models/UserModel.js";
import argon2 from "argon2";

export const getUsers = async(req, res) =>{
    try {
        const response = await User.findAll({
            attributes:['uuid','name','email','role']
        });
        res.status(200).json(response);
    } catch (error) {
        res.status(500).json({msg: error.message});
    }
}

export const getUserById = async(req, res) =>{
    try {
        const response = await User.findOne({
            attributes:['uuid','name','email','role'],
            where: {
                uuid: req.params.id
            }
        });
        res.status(200).json(response);
    } catch (error) {
        res.status(500).json({msg: error.message});
    }
}

export const createUser = async(req, res) =>{
    const {name, email, password, confPassword, role} = req.body;
    if(password !== confPassword) return res.status(400).json({msg:
"Password dan Confirm Password tidak cocok"});
    const hashPassword = await argon2.hash(password);
    try {
        await User.create({
            name: name,
            email: email,
```

```
            password: hashPassword,
            role: role
        });
        res.status(201).json({msg: "Register Berhasil"});
    } catch (error) {
        res.status(400).json({msg: error.message});
    }
}

export const updateUser = async(req, res) =>{
    const user = await User.findOne({
        where: {
            uuid: req.params.id
        }
    });
    if(!user) return res.status(404).json({msg: "User tidak ditemukan"});
    const {name, email, password, confPassword, role} = req.body;
    let hashPassword;
    if(password === "" || password === null){
        hashPassword = user.password
    }else{
        hashPassword = await argon2.hash(password);
    }
    if(password !== confPassword) return res.status(400).json({msg: "Password dan Confirm Password tidak cocok"});
    try {
        await User.update({
            name: name,
            email: email,
            password: hashPassword,
            role: role
        },{
            where:{
                id: user.id
            }
        });
        res.status(200).json({msg: "User Updated"});
    } catch (error) {
        res.status(400).json({msg: error.message});
    }
}

export const deleteUser = async(req, res) =>{
```

```
    const user = await User.findOne({
        where: {
            uuid: req.params.id
        }
    });
    if(!user) return res.status(404).json({msg: "User tidak
   ditemukan"});
    try {
        await User.destroy({
            where:{
                id: user.id
            }
        });
        res.status(200).json({msg: "User Deleted"});
    } catch (error) {
        res.status(400).json({msg: error.message});
    }
}
```

Untuk manajemen pengguna, fungsi `getUsers` dan `getUserById` digunakan untuk mengambil data pengguna secara keseluruhan atau berdasarkan ID tertentu. Fungsi `createUser` menangani pendaftaran pengguna baru dengan validasi password dan hashing menggunakan **argon2**. Sedangkan fungsi `updateUser` memperbarui data pengguna, termasuk opsi untuk mengubah password, dan `deleteUser` menghapus data pengguna dari sistem. Setiap operasi dilengkapi dengan validasi dan kontrol akses untuk memastikan keamanan dan integritas data.

5. Midleware/auth

```
import User from "../models/UserModel.js";

export const verifyUser = async (req, res, next) =>{
    if(!req.session.userId){
        return res.status(401).json({msg: "Mohon login ke akun Anda!"});
    }
    const user = await User.findOne({
        where: {
            uuid: req.session.userId
        }
    });
    if(!user) return res.status(404).json({msg: "User tidak
   ditemukan"});
    req.userId = user.id;
    req.role = user.role;
    next();
}
```

```
22.
23.export const adminOnly = async (req, res, next) =>{
24.    const user = await User.findOne({
25.        where: {
26.            uuid: req.session.userId
27.        }
28.    });
29.    if(!user) return res.status(404).json({msg: "User tidak
   ditemukan"});
30.    if(user.role !== "admin") return res.status(403).json({msg: "Akses
   terlarang"});
31.    next();
32.}
```

Kode ini bertujuan untuk mengatur middleware autentikasi dan otorisasi pengguna sebelum mereka mengakses fitur tertentu dalam aplikasi. Fungsi `verifyUser` digunakan untuk memverifikasi apakah pengguna telah login dengan memeriksa keberadaan sesi pengguna. Jika sesi tidak ditemukan atau pengguna tidak ada dalam database, permintaan akan dibatalkan dengan pesan error yang sesuai. Setelah verifikasi berhasil, middleware ini menambahkan informasi **userId** dan **role** ke objek `req` untuk digunakan di fungsi berikutnya, kemudian melanjutkan proses dengan memanggil fungsi `next()`. Fungsi `adminOnly` berperan sebagai middleware tambahan yang memastikan hanya pengguna dengan peran **admin** yang dapat mengakses rute tertentu. Setelah memverifikasi keberadaan pengguna, fungsi ini memeriksa apakah peran pengguna adalah **admin**. Jika tidak, akses ditolak dengan status 403 dan pesan "Akses terlarang." Jika pengguna memiliki peran yang sesuai, proses dilanjutkan dengan memanggil `next()`. Kedua fungsi ini mendukung implementasi kontrol akses berbasis peran, memastikan bahwa hanya pengguna yang memenuhi syarat yang dapat mengakses fitur-fitur tertentu dalam aplikasi.

6. Models/product models

```
7. import { Sequelize } from "sequelize";
8. import db from "../config/Database.js";
9. import Users from "./UserModel.js";
10.
11.const {DataTypes} = Sequelize;
12.
13.const Products = db.define('product',{
14.    uuid:{
15.        type: DataTypes.STRING,
16.        defaultValue: DataTypes.UUIDV4,
17.        allowNull: false,
18.        validate:{
19.            notEmpty: true
20.        }
```

```
    },
    name:{
        type: DataTypes.STRING,
        allowNull: false,
        validate:{
            notEmpty: true,
            len: [3, 100]
        }
    },
    price:{
        type: DataTypes.INTEGER,
        allowNull: false,
        validate:{
            notEmpty: true
        }
    },
    userId:{
        type: DataTypes.INTEGER,
        allowNull: false,
        validate:{
            notEmpty: true
        }
    }
},{
    freezeTableName: true
});

Users.hasMany(Products);
Products.belongsTo(Users, {foreignKey: 'userId'});

export default Products;
```

Model `Products` mendefinisikan tabel produk dengan kolom `uuid`, `name`, `price`, dan `userId`, yang mengaitkan setiap produk dengan pengguna tertentu melalui relasi. Relasi antara kedua tabel diatur dengan metode **hasMany** dan **belongsTo**, di mana satu pengguna dapat memiliki banyak produk. Dengan pengaturan ini, integritas data dan relasi antar entitas dalam database dapat terjaga.

7. Models/usermodel

```
import { Sequelize } from "sequelize";
import db from "../config/Database.js";

const {DataTypes} = Sequelize;

```

```
13.const Users = db.define('users',{
14.    uuid:{
15.        type: DataTypes.STRING,
16.        defaultValue: DataTypes.UUIDV4,
17.        allowNull: false,
18.        validate:{
19.            notEmpty: true
20.        }
21.    },
22.    name:{
23.        type: DataTypes.STRING,
24.        allowNull: false,
25.        validate:{
26.            notEmpty: true,
27.            len: [3, 100]
28.        }
29.    },
30.    email:{
31.        type: DataTypes.STRING,
32.        allowNull: false,
33.        validate:{
34.            notEmpty: true,
35.            isEmail: true
36.        }
37.    },
38.    password:{
39.        type: DataTypes.STRING,
40.        allowNull: false,
41.        validate:{
42.            notEmpty: true
43.        }
44.    },
45.    role:{
46.        type: DataTypes.STRING,
47.        allowNull: false,
48.        validate:{
49.            notEmpty: true
50.        }
51.    }
52.},{
53.    freezeTableName: true
54.});
55.
56.export default Users;
```

Program tersebut mendefinisikan model **Users**, menggunakan **Sequelize** sebagai ORM untuk mengelola tabel dalam database. Model `Users` merepresentasikan tabel pengguna dengan kolom seperti `uuid`, `name`, `email`, `password`, dan `role`, lengkap dengan validasi untuk memastikan data yang dimasukkan sesuai.

8. Routes/index

```
import express from "express";
import cors from "cors";
import session from "express-session";
import dotenv from "dotenv";
import db from "./config/Database.js";
import SequelizeStore from "connect-session-sequelize";
import UserRoute from "./routes/UserRoute.js";
import ProductRoute from "./routes/ProductRoute.js";
import AuthRoute from "./routes/AuthRoute.js";
dotenv.config();

const app = express();

const sessionStore = SequelizeStore(session.Store);

const store = new sessionStore({
    db: db
});

// (async()=>{
//     await db.sync();
// })();

app.use(session({
    secret: process.env.SESS_SECRET,
    resave: false,
    saveUninitialized: true,
    store: store,
    cookie: {
        secure: 'auto'
    }
}));

app.use(cors({
    credentials: true,
    origin: 'http://localhost:3000'
}));
app.use(express.json());
```

```
app.use(UserRoute);
app.use(ProductRoute);
app.use(AuthRoute);

// store.sync();

app.listen(process.env.APP_PORT, ()=> {
    console.log('Server up and running...');
});

```

Kode ini mengatur server menggunakan **Express** dengan fitur **session**, **CORS**, dan **dotenv** untuk konfigurasi. Server menggunakan **Sequelize** sebagai penyimpanan sesi melalui `connect-session-sequelize`, menghubungkannya dengan database. Middleware `session` dikonfigurasi untuk menyimpan sesi secara aman dengan kunci rahasia dari `.env`. **CORS** diatur agar hanya mengizinkan akses dari `http://localhost:3000`. Rute untuk pengguna, produk, dan autentikasi diimpor dari file terpisah dan digunakan pada server. Server berjalan di port yang ditentukan dalam file `.env` dan akan mencetak pesan ketika aktif.

9. Frontend main

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
```

```
29.       It will be replaced with the URL of the `public` folder during the
    build.
30.       Only files inside the `public` folder can be referenced from the
    HTML.
31.
32.       Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico"
    will
33.       work correctly both with client-side routing and a non-root public
    URL.
34.       Learn how to configure a non-root public URL by running `npm run
    build`.
35.     -->
36.     <title>React Redux App</title>
37.   </head>
38.   <body>
39.     <noscript>You need to enable JavaScript to run this app.</noscript>
40.     <div id="root"></div>
41.     <!--
42.       This HTML file is a template.
43.       If you open it directly in the browser, you will see an empty
    page.
44.
45.       You can add webfonts, meta tags, or analytics to this file.
46.       The build step will place the bundled scripts into the <body> tag.
47.
48.       To begin the development, run `npm start` or `yarn start`.
49.       To create a production bundle, use `npm run build` or `yarn
    build`.
50.     -->
51.   </body>
52. </html>
53.
```

File HTML ini adalah template dasar untuk aplikasi React yang dibuat menggunakan **Create React App (CRA)**. Template ini berfungsi sebagai struktur awal di mana aplikasi React akan dirender. Pada bagian `<head>`, terdapat elemen-elemen penting seperti pengaturan karakter encoding UTF-8, favicon, manifest, dan meta viewport untuk mendukung responsivitas. File `manifest.json` digunakan untuk konfigurasi Progressive Web App (PWA), sementara placeholder `%PUBLIC_URL%` memastikan akses file di folder `public` tetap konsisten saat build. Bagian `<body>` berisi elemen `<div id="root">`, tempat aplikasi React akan dimasukkan oleh React DOM. Selain itu, terdapat pesan di dalam tag `<noscript>` yang memberi tahu pengguna untuk mengaktifkan JavaScript. Template ini dirancang agar mendukung pengembangan dan build aplikasi dengan perintah seperti `npm start` untuk development dan `npm run build` untuk produksi.

IV. KESIMPULAN

Kesimpulannya, aplikasi ini dirancang menggunakan **Node.js**, **Express**, dan **Sequelize** dengan arsitektur multi-role untuk mengelola akses pengguna berdasarkan peran (admin dan user). Server dikonfigurasi untuk mendukung autentikasi berbasis sesi menggunakan **express-session** dan **Sequelize** sebagai penyimpanan sesi. Model **Users** dan **Products** didefinisikan dengan relasi one-to-many, memungkinkan setiap pengguna memiliki banyak produk. Middleware seperti `verifyUser` dan `adminOnly` memastikan kontrol akses yang ketat, memverifikasi pengguna yang login dan membatasi akses fitur tertentu hanya untuk admin.

Di sisi frontend, aplikasi React menggunakan file HTML template dari **Create React App** sebagai kerangka dasar tempat React merender komponen. Template ini mendukung Progressive Web App (PWA) dan diatur agar responsif, dengan elemen-elemen penting seperti meta tags dan manifest. Secara keseluruhan, aplikasi ini menerapkan kontrol akses yang aman, mendukung pengelolaan produk dan pengguna, serta siap untuk pengembangan dan produksi dengan proses build yang fleksibel.

Link Github:

https://github.com/maharaninelita/Login_multirole.git