Go-Green

Environmental Sustainability in our area to teach others on the importance of it.
and test
ya

Implementing Role-Based Authentication in a React and Node.js Application

1. Setup User Model

Ensure the User model includes a roles field to store user roles.

// models/User.js

```
const sequelize = require('../db'); // Assuming you have a Sequelize instance

const User = sequelize.define("User", {
    name: { type: DataTypes.STRING(50), allowNull: false },
    email: { type: DataTypes.STRING(50), allowNull: false, unique: true },
    password: { type: DataTypes.STRING(100), allowNull: false },
    roles: { type: DataTypes.STRING(100), allowNull: true } // Store roles as a
comma-separated string
});

User.associate = (models) => {
    User.hasMany(models.Tutorial, {
        foreignKey: "userId",
        onDelete: "cascade"
    });
};

module.exports = User;


# 2. Authentication Middleware
Create middleware to validate JWT tokens and check for admin roles.

// middlewares/auth.js
const { verify } = require('jsonwebtoken');
const { User } = require('../models');
require('dotenv').config();

const validateToken = (req, res, next) => {
  try {
    const accessToken = req.header("Authorization").split(" ")[1];
```

```
    if (!accessToken) {
      return res.sendStatus(401);
    }
    const payload = verify(accessToken, process.env.APP_SECRET);
    req.user = payload;
    return next();
  } catch (err) {
    return res.sendStatus(401);
  }
};

const checkAdminRole = async (req, res, next) => {
  const userId = req.user.id;
  const user = await User.findByPk(userId);
  if (user && user.roles.includes("ADMIN")) {
    return next();
  }
  return res.status(403).json({ message: "You do not have the required
permissions" });
};

module.exports = { validateToken, checkAdminRole };


# 3. Backend Routes
Use the authentication middleware in your routes to enforce role-based access.

// routes/user.js
const express = require('express');
const router = express.Router();
const bcrypt = require('bcrypt');
const { User } = require('../models');
const yup = require("yup");
const { sign } = require('jsonwebtoken');
const { validateToken } = require('../middlewares/auth');
require('dotenv').config();

router.post("/register", async (req, res) => {
    let data = req.body;
    let validationSchema = yup.object({
        name: yup.string().trim().min(3).max(50).required().matches(/^[a-zA-Z
'-,.]+$/, "Name only allows letters, spaces and characters: ' - , .'"),
        email: yup.string().trim().lowercase().email().max(50).required(),
        password: yup.string().trim().min(8).max(50).required().matches(/^(?=.*
[a-zA-Z])(?=.*[0-9]).{8,}$/, "Password must have at least 1 letter and 1
number")
    });
    try {
        data = await validationSchema.validate(data, { abortEarly: false });
        let user = await User.findOne({ where: { email: data.email } });
        if (user) {
            res.status(400).json({ message: "Email already exists." });
            return;
```

```
        }
        data.password = await bcrypt.hash(data.password, 10);
        let result = await User.create(data);
        res.json({ message: `Email ${result.email} was registered
successfully.` });
    } catch (err) {
        res.status(400).json({ errors: err.errors });
    }
});

router.post("/login", async (req, res) => {
    let data = req.body;
    let validationSchema = yup.object({
        email: yup.string().trim().lowercase().email().max(50).required(),
        password: yup.string().trim().min(8).max(50).required()
    });
    try {
        data = await validationSchema.validate(data, { abortEarly: false });
        let errorMsg = "Email or password is not correct.";
        let user = await User.findOne({ where: { email: data.email } });
        if (!user) {
            res.status(400).json({ message: errorMsg });
            return;
        }
        let match = await bcrypt.compare(data.password, user.password);
        if (!match) {
            res.status(400).json({ message: errorMsg });
            return;
        }
        let userInfo = {
            id: user.id,
            email: user.email,
            name: user.name,
            roles: user.roles
        };
        let accessToken = sign(userInfo, process.env.APP_SECRET, { expiresIn:
process.env.TOKEN_EXPIRES_IN });
        res.json({ accessToken: accessToken, user: userInfo });
    } catch (err) {
        res.status(400).json({ errors: err.errors });
        return;
    }
});

router.get("/auth", validateToken, (req, res) => {
    let userInfo = {
        id: req.user.id,
        email: req.user.email,
        name: req.user.name,
        roles: req.user.roles
    };
    res.json({ user: userInfo });
});
```

```
module.exports = router;

# 4. Role-Based Access in Announcement Routes
// routes/announcement.js
const express = require('express');
const router = express.Router();
const { validateToken, checkAdminRole } = require('../middlewares/auth');
const { Announcement } = require('../models');
const yup = require("yup");

router.put("/:id", validateToken, async (req, res) => {
    let id = req.params.id;
    let announcement = await Announcement.findByPk(id);
    if (!announcement) {
        res.sendStatus(404);
        return;
    }
    if (req.user.id === announcement.userId ||
req.user.roles.includes("ADMIN")) {
        let data = req.body;
        let validationSchema = yup.object({
            title: yup.string().trim().min(3).max(100),
            content: yup.string().trim().min(3).max(500),
            link: yup.string().trim().url(),
        });
        try {
            data = await validationSchema.validate(data, { abortEarly: false
});
            let num = await Announcement.update(data, { where: { id: id } });
            if (num == 1) {
                res.json({ message: "Announcement was updated successfully."
});
            } else {
                res.status(400).json({ message: `Cannot update announcement
with id ${id}.` });
            }
        } catch (err) {
            res.status(400).json({ errors: err.errors });
        }
    } else {
        res.status(403).json({ message: "You do not have the required
permissions" });
    }
});

router.delete("/:id", validateToken, async (req, res) => {
    let id = req.params.id;
    let announcement = await Announcement.findByPk(id);
    if (!announcement) {
        res.sendStatus(404);
        return;
    }
```

```
    if (req.user.id === announcement.userId ||
req.user.roles.includes("ADMIN")) {
        let num = await Announcement.destroy({ where: { id: id } });
        if (num == 1) {
            res.json({ message: "Announcement was deleted successfully." });
        } else {
            res.status(400).json({ message: `Cannot delete announcement with id
${id}.` });
        }
    } else {
        res.status(403).json({ message: "You do not have the required
permissions" });
    }
});

module.exports = router;

# 5. Frontend Implementation
Ensure the frontend correctly handles the user roles and displays UI elements
accordingly.

// components/Announcement.js
import React, { useEffect, useState, useContext } from "react";
import { Link } from "react-router-dom";
import {
  Box,
  Typography,
  Grid,
  Card,
  CardContent,
  Input,
  IconButton,
  Button,
} from "@mui/material";
import {
  AccountCircle,
  AccessTime,
  Search,
  Clear,
  Edit,
} from "@mui/icons-material";
import http from "../http";
import dayjs from "dayjs";
import UserContext from "../contexts/UserContext";
import HeaderWithBackground from "../components/HeaderWithBackground";
import global from "../global";

function Announcement() {
  const [announcementList, setAnnouncementList] = useState([]);
  const [search, setSearch] = useState("");
  const { user } = useContext(UserContext);

  const onSearchChange = (e) => {
```

```
    setSearch(e.target.value);
  };

  const getAnnouncements = () => {
    http.get("/announcement").then((res) => {
      setAnnouncementList(res.data);
    });
  };

  const searchAnnouncement = () => {
    http.get(`/announcement?search=${search}`).then((res) => {
      setAnnouncementList(res.data);
    });
  };

  const onSearchKeyDown = (e) => {
    if (e.key === "Enter") {
      searchAnnouncement();
    }
  };

  const onClickSearch = () => {
    searchAnnouncement();
  };

  const onClickClear = () => {
    setSearch("");
    getAnnouncements();
  };

  useEffect(() => {
    getAnnouncements();
  }, []);

  return (
    <Box>
      <HeaderWithBackground title="Announcements"
backgroundImage="/uploads/test.jpg" />

      <Box sx={{ display: "flex", alignItems: "center", mb: 2 }}>
        <Input value={search} placeholder="Search" onChange={onSearchChange}
onKeyDown={onSearchKeyDown} />
        <IconButton color="primary" onClick={onClickSearch}>
          <Search />
        </IconButton>
        <IconButton color="primary" onClick={onClickClear}>
          <Clear />
        </IconButton>
        <Box sx={{ flexGrow: 1 }} />
        <Link to="/addannouncement" style={{ textDecoration: "none" }}>
          <Button variant="contained">Add</Button>
        </Link>
      </Box>
```

```
      <Grid container spacing={2}>
        {announcementList.map((announcement) => (
          <AnnouncementCard key={announcement.id} announcement={announcement}
user={user} />
        ))}
      </Grid>
    </Box>
  );
}

function AnnouncementCard({ announcement, user }) {
  const [isExpanded, setIsExpanded] = useState(false);

  const toggleExpanded = () => {
    setIsExpanded(!isExpanded);
  };

  return (
    <Grid item xs={12} md={6} lg={4}>
      <Card>
        {announcement.imageFile && (
          <Box className="aspect-ratio-container">
            <img alt="announcement" src=
{`${import.meta.env.VITE_FILE_BASE_URL}${announcement.imageFile}`} />
          </Box>
        )}
        <CardContent>
          <Box sx={{ display: "flex", mb: 1 }}>
            <Typography variant="h6" sx={{ flexGrow: 1 }}>
              {announcement.title || "No Title"}
            </Typography>
            {(user?.roles?.includes("ADMIN") || user?.id ===
announcement.userId) && (
              <Link to={`/editannouncement/${announcement.id}`}>
                <IconButton color="primary" sx={{ padding: "4px" }}>
                  <Edit />
                </IconButton>
              </Link>
            )}
          </Box>
          <Box sx={{ display: "flex", alignItems: "center", mb: 1 }}
color="text.secondary">
            <AccountCircle sx={{ mr: 1 }} />
            <Typography>{announcement.user?.name || "Unknown User"}
</Typography>
          </Box>
          <Box sx={{ display: "flex", alignItems: "center", mb: 1 }}
color="text.secondary">
            <AccessTime sx={{ mr: 1 }} />
            <Typography>
              {announcement.createdAt ?
dayjs(announcement.createdAt).format(global.datetimeFormat) : "Unknown Date"}
            </Typography>
```

```
          </Box>
          <Typography sx={{ whiteSpace: "pre-wrap", pb: 2 }}>
            {isExpanded
              ? announcement.content
              : `${announcement.content?.substring(0,
500)}${announcement.content?.length > 500 ? "..." : ""}` || "No Content"}
          </Typography>

          {announcement.link && (
            <Typography>
              Link:
              <Box component="span" sx={{ ml: 1 }}>
                <a href={announcement.link} target="_blank" rel="noopener
noreferrer">
                  {announcement.link}
                </a>
              </Box>
            </Typography>
          )}
        </CardContent>
      </Card>
    </Grid>
  );
}

export default Announcement;```
```