For Nodejs

Authorize Role

Step 1: Create the Middleware Create a new file authorizeRole.js inside your middleware directory or at the root of application and add the following code:

```
const jwt = require("jsonwebtoken");
// import your secret key here
const config = require("../config");

// Middleware for role authorization
const authorizeRole = (allowedRoles) => {
  return (req, res, next) => {
    // Get the JWT token from the request header
    const token = req.headers.authorization.split(" ")[1];
    console.log("AR token ", token);

    try {
      // Decode the JWT token to extract user information
      const decoded = jwt.verify(token, config.SECRET_KEY);
      console.log("AuthorizeRole decoded ", decoded);

      const userRole = decoded.role; // Assuming role is stored in the decoded
token
      console.log("userRole ", userRole);

      // Check if the user's role is included in the allowed roles
      if (allowedRoles.includes(userRole)) {
        // User has the necessary role, allow access to the route
        next();
      } else {
        // User does not have the necessary role, return a 403 Forbidden error
        return res.status(403).json({ error: "Access forbidden" });
      }
    } catch (error) {
      // Handle token verification error (e.g., token expired, invalid token)
      return res.status(401).json({ error: "Unauthorized" });
    }
  };
};

module.exports = authorizeRole;
```

Step 2: Apply Middleware to Routes Open the route files where you want to apply role-based authorization and import the middleware. Use the authorizeRole function to protect routes.

```
// unprotected route
// GET ALL STAFF MEMBERS
```

```
router.get("/all-staff", (request, response) => {
  const statement = `SELECT * FROM ${STAFF_TABLE}`;

  db.execute(statement, (error, result) => {
    response.send(utils.createResponse(error, result));
  });
});
------------------------------------------------------
// make it as protected route
// here a user with role = admin will be allowed to access this route
// if you want to allow access to other user
// then add roles like this  authorizeRole(["admin","coordinator", "staff"])
// GET ALL STAFF MEMBERS
router.get("/all-staff", authorizeRole(["admin"]), (request, response) => {
  const statement = `SELECT * FROM ${STAFF_TABLE}`;

  db.execute(statement, (error, result) => {
    response.send(utils.createResponse(error, result));
  });
});
```

For Flask

Step 1: Create the Middleware Create a new file auth.py at the root of application and add the following code:

```
# auth.py
# functools- Used to preserve the original function's metadata when decorating it.
from functools import wraps
from flask import request, jsonify
import jwt
from config import SECRET_KEY
from utils import create_error_response

# Define the authorize_role function which takes allowed roles as an argument
def authorize_role(allowed_roles):
    def decorator(f):
        @wraps(f)
        def decorated_function(*args, **kwargs):
            # Get the Authorization header from the request
            auth_header = request.headers.get("Authorization")
            if not auth_header:
                return jsonify(create_error_response("missing authorization
header")), 401

            token = auth_header.split(" ")[1]
            if not token:
                return jsonify(create_error_response("missing token")), 401

            try:
                payload = jwt.decode(token, SECRET_KEY, algorithms=["HS256"])
```

```
                print("Decoded Token:", payload)  # Print the decoded token
                user_role = payload.get("role")

                if user_role not in allowed_roles:
                    # If the user's role is not in the allowed roles, return a 403
error
                    return jsonify(create_error_response("Access forbidden")), 403
            except jwt.ExpiredSignatureError:
                return jsonify(create_error_response("expired token")), 401
            except jwt.InvalidTokenError:
                return jsonify(create_error_response("invalid token")), 401

            return f(*args, **kwargs)
        return decorated_function
    return decorator
```

Step 2: Import the authorize_role in the routes files

```
from flask import Blueprint, jsonify
from auth import authorize_role
from db import get_db_connection
from utils import create_success_response, create_error_response

staff_bp = Blueprint('staff', __name__)

# Protected Route
# add @authorize_role(['admin'])
# to pass multiple roles use @authorize_role(['admin','coordinator'])
@staff_bp.route("/all-staff", methods=["GET"])
@authorize_role(['admin'])
def get_all_staff():
    try:
        conn = get_db_connection()
        cursor = conn.cursor()

        statement = "SELECT * FROM staff"
        cursor.execute(statement)
        result = cursor.fetchall()

        return jsonify(create_success_response(result))
    except Exception as e:
        return jsonify(create_error_response(str(e)))
    finally:
        cursor.close()
        conn.close()
```
