Experiment 6.3: Build Role-Based Access Control with Admin, User, and Moderator Roles

Objective:

To implement Role-Based Access Control (RBAC) in a Node.js application using JWT authentication, where different roles like Admin, User, and Moderator have specific permissions.

Theory:

Role-Based Access Control (RBAC) is a method to restrict system access based on user roles. Each role defines a set of permissions determining what operations a user can perform. In this experiment, JWT is used to identify a user and their role, and middleware is applied to check access levels before processing requests.

Code Implementation:

```
const express = require('express');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');
require('dotenv').config();

const app = express();
app.use(express.json());
const PORT = process.env.PORT || 5000;

const users = [
  { id: 1, username: "admin", role: "admin", password: "$2a$10$Qxz/jav5Hrq8Ds5HddQHve2eqGJmDYgZCsy5t
  { id: 2, username: "moderator", role: "moderator", password: "$2a$10$Qxz/jav5Hrq8Ds5HddQHve2eqGJmD
  { id: 3, username: "user", role: "user", password: "$2a$10$Qxz/jav5Hrq8Ds5HddQHve2eqGJmDYgZCsy5tz8
];

app.post('/login', async (req, res) => {
  const { username, password } = req.body;
  const user = users.find(u => u.username === username);
  if (!user) return res.status(400).json({ message: "User not found" });

  const isMatch = await bcrypt.compare(password, user.password);
  if (!isMatch) return res.status(401).json({ message: "Invalid credentials" });

  const token = jwt.sign(
    { id: user.id, username: user.username, role: user.role },
    process.env.JWT_SECRET || "mysupersecretkey",
    { expiresIn: '1h' }
  );

  res.json({ message: "Login successful", token });
});

const verifyToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];
  if (!token) return res.status(403).json({ message: "Access denied. No token provided." });

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET || "mysupersecretkey");
    req.user = decoded;
    next();
  } catch (err) {
    res.status(401).json({ message: "Invalid or expired token" });
  }
};

const authorizeRoles = (...roles) => {
  return (req, res, next) => {
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({ message: "Access forbidden: insufficient rights" });
    }
    next();
  };
};

app.get('/admin', verifyToken, authorizeRoles('admin'), (req, res) => {
  res.json({ message: `Welcome Admin ${req.user.username}` });
});

app.get('/moderator', verifyToken, authorizeRoles('admin', 'moderator'), (req, res) => {
  res.json({ message: `Hello Moderator ${req.user.username}` });
});

app.get('/user', verifyToken, authorizeRoles('admin', 'moderator', 'user'), (req, res) => {
  res.json({ message: `Welcome User ${req.user.username}` });
});

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

Learning Outcome:

By performing this experiment, students learn how to implement Role-Based Access Control (RBAC) using JWTs in Node.js. They understand how to assign roles to users and restrict access to certain routes based on these roles.