Experiment 6.3

Aim

To implement role-based access control (RBAC) in a Node.js and Express application, restricting access to certain routes based on user roles.

Introduction

Role-Based Access Control (RBAC) is a security paradigm where access permissions are assigned based on user roles. This approach ensures that only authorized users can access certain functionalities. In this experiment, JWT tokens are used to store user roles, and middleware functions are implemented to verify tokens and enforce role-specific access.

Procedure

1. Initialize a Node.js project and install necessary packages: `express`, `jsonwebtoken`, and `body-parser`.
2. Create sample users with different roles (Admin, User, Moderator) for testing.
3. Implement a login route that issues JWT tokens containing the user's role in the payload.
4. Create a middleware `verifyToken` to check the JWT and extract the user's role.
5. Implement an `authorizeRole` middleware to restrict access based on user roles.
6. Create protected routes:
7. Admin-only dashboard route.
8. Moderator management route.
9. General User profile route.
10. Ensure that requests with invalid tokens or insufficient roles are denied with appropriate error messages.
11. Test the system by logging in with different roles and accessing each route to confirm RBAC functionality.

Code

server.js

```
const express = require('express');
const jwt = require('jsonwebtoken');
const bodyParser = require('body-parser');

const app = express();
app.use(bodyParser.json());

const PORT = 5000;
const SECRET_KEY = 'rbacsecret';
```

```
// Sample users
const users = [
  { username: 'admin', password: 'admin123', role: 'Admin' },
  { username: 'moderator', password: 'mod123', role: 'Moderator' },
  { username: 'user', password: 'user123', role: 'User' }
];

// Login Route
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  const user = users.find(u => u.username === username && u.password ===
password);
  if (!user) return res.status(401).json({ message: 'Invalid credentials' });

  const token = jwt.sign({ username: user.username, role: user.role },
SECRET_KEY, { expiresIn: '1h' });
  res.json({ token });
});

// JWT Verification Middleware
const verifyToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  if (!authHeader) return res.status(401).json({ message: 'Authorization header
missing' });
  const token = authHeader.split(' ')[1];
  jwt.verify(token, SECRET_KEY, (err, user) => {
    if (err) return res.status(403).json({ message: 'Invalid token' });
    req.user = user;
    next();
  });
};

// Role Authorization Middleware
const authorizeRole = (roles) => (req, res, next) => {
  if (!roles.includes(req.user.role)) return res.status(403).json({ message:
'Access denied' });
  next();
};

// Routes
app.get('/admin-dashboard', verifyToken, authorizeRole(['Admin']), (req, res)
=> {
  res.json({ message: 'Welcome to the Admin Dashboard' });
});

app.get('/moderator-panel', verifyToken, authorizeRole(['Moderator']), (req,
res) => {
```

```
  res.json({ message: 'Welcome to the Moderator Panel' });
});

app.get('/user-profile', verifyToken, authorizeRole(['User', 'Admin',
'Moderator']), (req, res) => {
  res.json({ message: `Welcome ${req.user.username} to your profile` });
});

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

Output

- **Login Route**

```
Request Body: { "username": "admin", "password": "admin123" }
Response: { "token": "<JWT_TOKEN>" }
```

- **Protected Routes with Valid Role**

```
GET /admin-dashboard with Admin token
Response: { "message": "Welcome to the Admin Dashboard" }

GET /moderator-panel with Moderator token
Response: { "message": "Welcome to the Moderator Panel" }

GET /user-profile with any valid token
Response: { "message": "Welcome <username> to your profile" }
```

- **Protected Routes with Invalid Role or Token**

```
Response: { "message": "Access denied" } or { "message": "Invalid token" }
```

Learning Outcomes

- Understand and implement role-based access control in a Node.js and Express application.
- Learn to store and check user roles in JWT tokens.
- Create flexible authorization middleware for multiple user roles.
- Secure backend API routes against unauthorized access.
- Test the system thoroughly to ensure role-based restrictions are enforced.